Università di Pisa

Facoltà di Ingegneria

Corso di Laurea Specialistica in

Ingegneria Informatica per la Gestione d'Azienda

Progettazione e sviluppo di una Java Enterprise Application per la gestione del Total Cost of Ownership di una multinazionale leader nel settore dei trasporti

Tesi di Laurea di

Michele Bursi

09 Maggio 2013

Relatori:

Prof. Enzo Mingozzi

Prof. Roberto Chiavaccini

Dr. Uwe Steffen

Anno Accademico 2012 - 2013

ai miei genitori, Carla e Marcello

Indice

Elenco delle figure

Capitolo 1

Introduzione

Il lavoro di tesi che viene presentato è stato svolto presso Bombardier, più precisamente nel quartier generale del settore Transportation sito a Berlino, Germania.

Bombardier[1] è una multinazionale canadese, fondata nel 1942 a Valcourt nel Québec, l'attuale sede si trova a Montréal. L'attività primaria dell'azienda è relativa ai settori dei trasporti (*Bombardier Transportation*) e dell'aeronautica (*Bombardier Aerospace*).

Bombardier è uno dei quattro maggiori produttori aeronautici di linea del mondo con Boeing, Airbus ed Embraer, e il primo produttore planetario di velivoli e aerei regionali; è inoltre la terza industria mondiale per forza lavoro. Per ciò che riguarda gli altri settori Bombardier è prima nella costruzione di veicoli spaccaghiaccio, spazzaneve, trivelle, macchinari da lavoro tra i ghiacci e veicoli militari. Inoltre è l'azienda costruttrice di gran parte delle metropolitane del mondo, fra cui quella di Montreal (dove ha sede l'azienda) e la DLR di Londra. Bombardier è l'azienda produttrice degli aerei Canadair (chiaramente da Canada). In Italia, nel 2001, ha acqui-

[1] http://www.bombardier.com/

sito la DaimlerChryslerAdtranz di Vado Ligure costituendo la Bombardier Transportation Italy, specializzata nella costruzione di rotabili e impianti ferroviari[1]. L'azienda ha inoltre altre attività:

- *Bombardier Capital*
- *Bombardier Real Estate Services*
- *Bombardier Recreational Products*:
  - Snowmobiles (Ski-Doo e Lynx)
  - Sea-Doo (watercraft e sport boats)
  - Can-Am/Atv/Quad
  - Motori marini (Evinrude e Johnson)
  - Rotax
  - Kart

Bombardier Transportation è inoltre uno dei principali produttori di sistemi per il controllo del traffico ferroviario[2].

Le esigenze, in termini di sistemi ed infrastrutture informatiche, di una multinazionale presente in più di 60 paesi nel mondo, con 76 siti produttivi e di ingegneria ed oltre 70000 mila dipendenti, sono decisamente un fattore non trascurabile. Tutti i sistemi e le infrastrutture necessarie devono essere gestite in maniera attenta e rigorosa e chiaramente il controllo dei costi di questi sistemi ed infrastrutture è di primaria importanza per una corretta gestione.

[2]European Rail Traffic Management System/European Train Control System ERTMS/ETCS

L'azienda ha deciso di adottare il modello Total Cost of Ownership (TCO) sviluppato da Gartner[3] nel 1987, utilizzato per calcolare tutti i costi del ciclo di vita di un'apparecchiatura informatica IT, per l'acquisto, l'installazione, la gestione, la manutenzione e il suo smantellamento.

Viste le enormi esigenze informatiche dell'azienda in questione, la gestione dei dati necessari all'utilizzo del TCO sono di vitale importanza e l'azienda ha la necessità di gestire tali dati con la massima cautela, ma al tempo stesso ha la necessità di dover fornire un metodo univoco per l'accesso a tali dati, rispettando vincoli di integrità, sicurezza, autenticazione e disponibilità.

All'esigenza di poter fornire un metodo per poter gestire tali dati si affiancano le classiche problematiche di integrazione in un ambiente enterprise totalmente distribuito geograficamente. Fin dal principio è inoltre apparso evidente che, per gestire i dati generati dall'approccio di Gartner fosse necessario rispettare vari vincoli che rendono l'utilizzo di tecnologie *enterprise* indispensabile. Le tecnologie enterprise sono infatti diventate negli anni sinonimo di sviluppo di applicazioni aziendali robuste, sicure ed efficienti, permettendo inoltre la creazione di nuovi modelli B2B (Business to Business) e B2C (Business to Consumer) che facilitano le aziende nello sviluppo di servizi innovativi. E' inoltre importante sottolineare che tali tecnologie, nelle realtà aziendali attuali, non rappresentano solo uno strumento ma bensì un'opportunità di ottenere dei vantaggi competitivi di elevata importanza strategica.

Dopo un'attenta analisi di varie soluzioni tecnologiche presenti sul mercato, lo sviluppo di due differenti prototipi di un'applicazione, è stato quindi deciso di utilizzare i framework messi a disposizioni per il linguaggio Java, in quanto risultano essere tra i più usati, stabili e sviluppati in ambito

[3]http://it.wikipedia.org/wiki/Gartner

enterprise. La tesi presenta la seguente struttura:

- Nel Capitolo 2 verrà introdotto il problema presente in azienda, l'approccio TCO, le esigenze dall'azienda e i requisiti che l'applicazione deve rispettare.

- Nel Capitolo 3 verrà fornita un'analisi dello stato dell'arte dei framework Java per Enterprise Application attualmente presenti nel mercato, mettendone in luce pregi e difetti, verrà inoltre introdotta un'altra tecnologia basata sul linguaggio Ruby e sul framework Ruby On Rails.

- Nel Capitolo 4 saranno illustrati i due prototipi sviluppati inizialmente, uno con linguaggio Ruby e un web framework molto utilizzato in tale tecnologia, l'altro con la tecnologia Java scelta dopo l'analisi dei framework affrontata nel capitolo precedente. Dopodiché sarà presentata l'architettura che è stata individuata come la più idonea a risolvere il problema proposto. In particolare saranno illustrate le tecnologie che sono state scelte, rispetto a quelle presentate nel capitolo precedente, e come permettano di rispettare i requisiti.

- Nel Capitolo 5 verrà presentato il prototipo che è stato sviluppato al fine di verificare il funzionamento dell'architettura proposta.

- Infine, nel Capitolo 6, verranno illustrate le conclusioni ottenute.

Capitolo 2

Presentazione del problema

L'azienda presso cui è stata svolta la tesi si occupa di progettare e realizzare tutto ciò che concerne al mondo dei trasporti rotabili. I servizi informatici ricoprono a tutto tondo tutte le aree dell'azienda e tutte le attività primarie e di supporto che possiamo trovare nella catena del valore di Porter.

Quindi non solo servizi informatici per le attività core dell'azienda, ma servizi globali che vanno ad abbracciare qualsiasi sfaccettatura dell'ambiente enterprise. Questo fa facilmente intendere che le necessità in termini di risorse, applicazioni e software che servono per coprire tutte le esigenze di una multinazionale geograficamente dislocata in più di 60 paesi sono decisamente elevate. La capacità di mantenere un'infrastruttura ed un portafoglio applicativo vario, efficace, efficiente e che copra totalmente tutte le necessità che possono aver luogo in tale realtà, è un aspetto di gestione non trascurabile.

L'azienda già da alcuni anni ha adottato ed implementato il modello TCO per quanto riguarda tutto ciò che riguarda l'ambito IT. L'esigenza nasce dal motivo di avere una stima economica attendibile che possa aiutare clienti enterprise manager a determinare costi diretti e indiretti legati all'ambito IT.

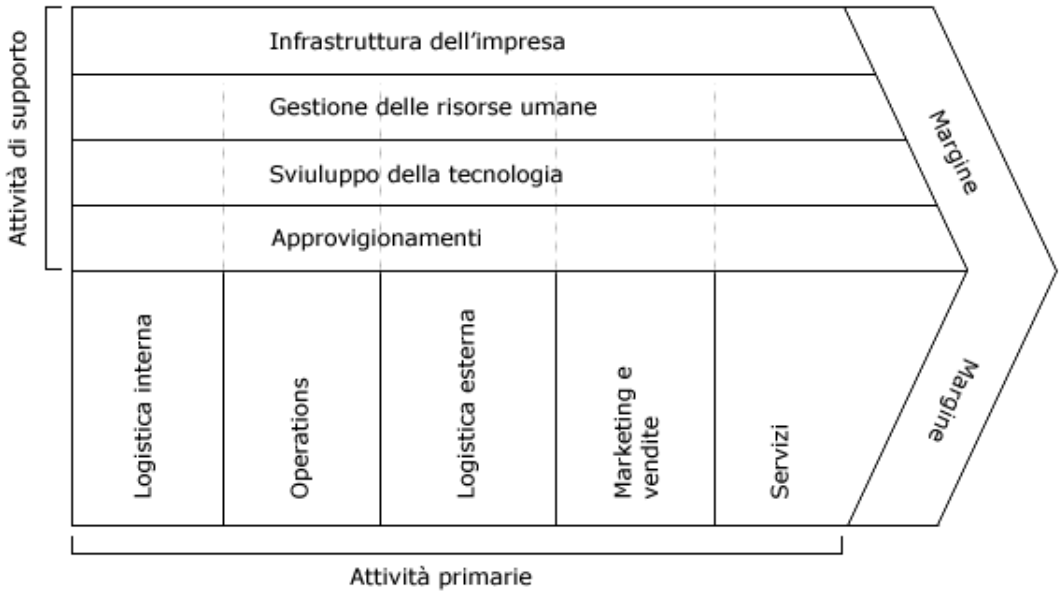

Figura 2.1: Catena del Valore di Porter

Lo scopo della tesi è stato quello di progettare e sviluppare un'applicazione enterprise per la gestione di un database già presente, database che contenente tutte le voci per la corretta gestione delle apparecchiature IT secondo l'approccio TCO. Per sviluppare tale enterprise application sono stati presi in considerazione vari aspetti, da quelli prettamente tecnici, a quelli di integrazione all'interno dell'ambiente enterprise preesistente ad aspetti meno tecnici ma strettamente legati all'utilizzo dell'applicazione stessa all'interno dell'organizzazione, poiché l'utilizzo finale abbracciava un bacino di utenza con livelli di conoscenza tecnica differenti e quindi anche con capacità di interazione differente con sistemi informatici.

In questo capitolo sarà presentato il modello TCO, i requisiti e le esigenze dell'azienda per quanto riguarda lo sviluppo delle Java Enterprise Application per la gestione ed il corretto funzionamento dell'approccio TCO

2.1 TCO - Total Cost of Ownership

L'analisi TCO è stata resa popolare dal Gartner Group[1] nel 1987. La radice di questo concetto però è molto più vecchia e risale al primo quarto del ventesimo secolo. Questo approccio era già in uso nell'ambito dei beni durevoli di largo consumo quali macchinari industriali, mezzi di trasporto e veicoli in genere. Lo sviluppo nel settore IT è stato determinato dalla velocissima obsolescenza hardware e software e dalla necessità da parte delle grandi aziende di avere dei riferimenti di costo attendibili. Proprio per questo i vari processi di *Make or Buy* sono stati trasferiti al concetto di *Buy or Rent*[2]. L'analisi TCO deve tener conto di vari tipi di costi:

- costi per l'acquisto dei componenti hardware o software
- costi per lo sviluppo di personalizzazione degli applicativi
- costi operativi, legati all'aggiornamento e alla manutenzione e all'esercizio del software
- costi legati alla dismissione del sistema

Più precisamente come parte dell'analisi TCO viene tenuto conto di:

- Hardware and software
  - Network hardware/software
  - Server hardware/software
  - Workstation hardware/software
  - Installazione e integrazione di hardware e software
  - Ricerca prodotti

[1] http://www.gartner.com/

  - Garanzie e licenze
  - Tracciamento licenze e conformità
  - Costi di migrazione
  - Rischi: vulnerabilità, disponibilità di aggiornamenti, patches, termini di licenza futuri, ecc.

- Costi operativi
  - Infrastrutture
  - Elettricità
  - Costi di testing
  - Downtime, interruzione e costi di guasto
  - Diminuzione delle performance: utenti che devono attendere si traduce in minore capacità di monetizzare
  - Sicurezza
  - Costi di backup e recovery
  - Costi di training
  - Ispezioni interne ed esterne
  - Assicurazione
  - Personale IT
  - Tempo di gestione da parte dell'azienda

- Costi di lungo termine
  - Costi di rimpiazzamento
  - Costi di upgrade futuri e/o spese di scalabilità

– Costi di decommissionamento

L'approccio TCO è basato sul fatto che il costo totale di una apparecchiatura IT non dipende solo dai costi di acquisto, ma anche da tutti i costi che intervengono durante l'intera vita di esercizio. I costi direttamente connessi all'utilizzo del TCO sono abbastanza rilevanti ed in ogni modo considerati nell'analisi stessa. Questo però porta il fatto che questo approccio viene spesso utilizzato solo da grandi aziende in cui la variabile IT rappresenta un fattore strategico rilevante.

Il TCO essendo un'analisi stati dei costi di esercizio di una apparecchiatura, non tiene in considerazione eventuali ritorni dovuti all'investimento in innovazione, del risparmio operativo e delle nuove mansioni create. E' sicuramente un metodo ottimo per misurare i costi totali, identificando tutte le spese in termini chiari e misurabili, ma presenta il problema di non poter dare una stima del valore complessivo aziendale.

2.2 User interface design

L'approccio TCO in azienda è stato adottato alcuni anni fa, all'inizio del lavoro svolto era quindi già presente una notevole mole di dati all'interno del database. Chiaramente l'interazione con un Relational Database Management System (RDBMS) direttamente con tool specifici o da linea di comando non è di facile utilizzo e quanto meno richiede delle conoscenze tecniche particolari, conoscenze solitamente a disposizione di personale propriamente formato. Visto che l'utilizzo dell'approccio TCO all'interno dell'azienda però coinvolge un insieme di persone con conoscenze eterogenee, tecniche e non tecniche, già in passato si è sentita l'esigenza di fornire un interfaccia esterna per l'interazione e la gestione dei dati contenuti all'interno del database.

Con il passare del tempo il database è stato fortemente ampliato, nuove tabelle sono state aggiunte per adempiere a nuovi compiti e l'interfaccia creata inizialmente a iniziato a diventare sempre più complessa e meno semplice da utilizzare. Anche la scelta di tecnologie non propriamente adatte ad adempiere a tale compito a portato ad un notevole degrado dell'esperienza utente e della semplicità d'uso di tale interfaccia, fino ad arrivare a renderla molto complessa e di difficile comprensione per nuovi dipendenti che si trovavano a doverla utilizzare. Inoltre la tecnologia scelta non permetteva un concorrente utilizzo del software ed erano necessari continui aggiornamenti con versioni più recenti del software. Questo ulteriore ostacolo ha fatto crescere l'esigenza di avere un'applicazione enterprise che integrasse le funzionalità dell'interfaccia esistente ma riprogettando da zero l'interfaccia stessa in un ottica di migliore interazione con l'utente e semplicità di utilizzo.

2.3 Requisiti dell'applicazione

2.3.1 Integrità dei dati

Chiaramente il valore principale dell'applicazione sono i dati contenuti all'interno del database TCO. Preservare questi dati, trattarli correttamente, verificarne la correttezza a seguito di input utente è risultato uno dei fattori principali di cui tener conto nella progettazione e sviluppo dell'applicazione.

2.3.2 Affidabilità

Il sistema in oggetto ha la necessità di fornire risposte consistenti anche in caso di guasto e preservare l'integrità dei dati anche in situazioni anomale. E'

stato necessario prevedere meccanismi di fault tolerance per evitare perdite di dati o immissione di dati errati nel database.

2.3.3 Disponibilità

L'interazione con il database all'interno dell'azienda è un attività continua e necessaria. Rendere il servizio sempre disponibile è risultato essere un requisito di cui tener conto, questo ha portato a pensare a meccanismi di clustering del sistema in produzione.

2.3.4 Manutenibilità

Vista la veloce evoluzione delle esigenze di una multinazionale, che coinvolgono pure molti aspetti legati al database TCO, è risultato di fondamentale importanza fornire un applicazione ben strutturata con un livello di manutenibilità elevato. Facile da estendere ad adattare ad esigenze future e a cambiamenti necessari dovuti alle modiche apportate alla struttura del database TCO

2.3.5 Integrazione in ambiente Enterprise

Altro requisito fondamentale è stato quello di integrare il sistema con l'ambiente circostante, per quanto riguarda i vari database e i servizi di autenticazione enterprise.

2.3.6 Controllo degli accessi

La versione precedente dell'applicazione non prevedeva nessun controllo degli accessi ne meccanismi di controllo di alcun tipo su operazioni eseguite sull'in-

terfaccia stessa (che poi venivano eseguite nel database stesso). E' risultato chiaro che un requisito del genere fosse di elevata importanza.

2.3.7 Interazione con sistemi esistenti

Fin dall'inizio si è voluto prevedere che altri sistemi nell'ambiente enterprise potessero interrogare l'applicazione per ricevere dati. Si è voluto progettare l'applicazione come middleware tra l'RDBMS e altri sistemi.

2.4 AS IS

L'AS IS del sistema all'inizio era composto da un RDBMS Microsoft SQL Server 2005 contenente tutti i dati dell'approccio TCO. Un'applicazione sviluppata in Microsoft Access che forniva un interfaccia per l'accesso al database TCO. Nessun meccanismo di controllo degli accessi, meccanismi di correttezza dei dati minimi e facenti riferimento a funzionalità del RDBMS.

Capitolo 3

Analisi comparativa dei framework per Enterprise Application

L'azienda aveva interesse a sviluppare l'attuale applicazione in una delle tre tecnologie:

- Java
- Ruby
- .NET

Le mie precedenti esperienze lavorative ed accademiche mi hanno fornito un background nelle prime due tecnologie elencate: Java e Ruby. Questo ha portato me e il mio team a focalizzarci alla ricerca della miglior framework che avrebbe consentito di rispettare tutti i requisiti dell'azienda. In questo capitolo verranno illustrati tutti i framework e le soluzioni prese in considerazione, sia per quanto riguarda Ruby che Java. Verranno illustrati caratteristiche peculiari di ognuno di essi e giustificate le scelte effettuate.

3.1 Java

3.1.1 Java Enterprise Edition (JavaEE v6)

La tecnologia Java Enterprise Edition (JavaEE)[3] è diventata negli anni sinonimo di sviluppo di applicazioni aziendali robuste, sicure ed efficienti. Queste caratteristiche la rendono tra le più importanti piattaforme tecnologiche di sviluppo, soprattutto in ambiti in cui la sicurezza e la robustezza sono vincoli imprescindibili (ad esempio applicazioni bancarie).

La tecnologia JavaEE può facilitare la creazione di modelli Business to Business (B2B) e Business to Consumer (B2C) e quindi permettere all'azienda lo sviluppo di nuovi servizi. Infatti, attraverso il modello di sviluppo proposto, rende facile l'accesso ai dati e la sua rappresentazione in diverse forme (un browser web, un applet, un dispositivo mobile, un sistema esterno, ecc).

Dal punto di vista tecnologico tutto ciò è realizzato con una struttura a livelli (*layer*), dove ogni livello implementa uno specifico servizio, a partire dal quale può essere implementato il processo aziendale. JavaEE è dunque un raccoglitore di tecnologie che facilitano lo sviluppo di software web based distribuito (Figura 3.1).

La specifica JavaEE prevede un'architettura a *layer* ed in particolare viene suddivisa in:

- Business Layer
- Presentation Layer

Il Business Layer si occupa di effettuate tutte le operazioni riguardanti la *business logic*, ovvero sono presenti le funzioni che si occupano di elaborare i dati ed interagire con database, Enterprise Information System (EIS)

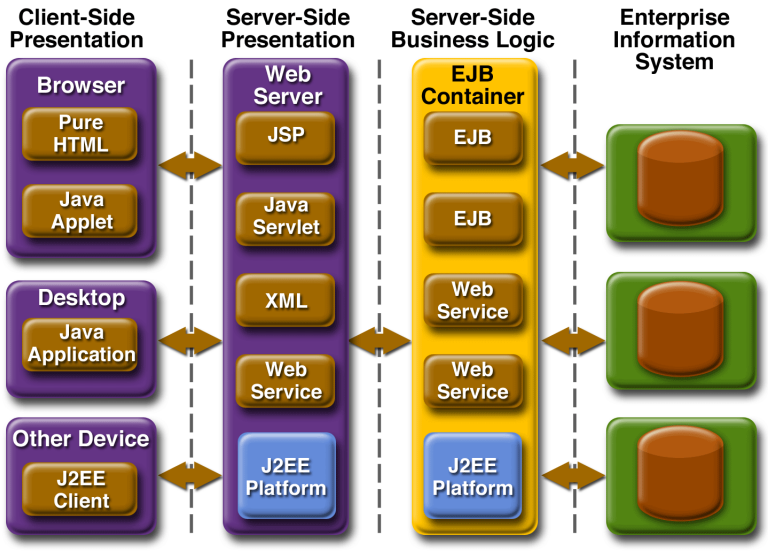

Figura 3.1: Architettura JavaEE

o sistemi *legacy*. Il business layer ha anche il compito di fornire i dati al presentation layer in modo che possano essere visualizzati all'utente.

Compito del Presentation Layer è quindi quello di richiedere i dati al Business Layer e trasformarli in formato che permetta di presentarli agli utenti. Allo stesso tempo, il presentation layer, si occuperà anche di ricevere degli *input* dagli utenti e comunicarli al business layer in maniera che possano essere eseguite le operazioni richieste.

E' da notare che questa divisione è puramente logica, ovvero più layer possono risiedere sullo stesso server o su server diversi. In seguito sarà specificato come può essere effettuata tale separazione.

Per poter realizzare applicazioni Java Enterprise è necessario disporre di un *application server*, ovvero di un software che fornisca l'infrastruttura e le funzionalità di supporto, sviluppo ed esecuzione di applicazioni e componenti server in un contesto distribuito. Solitamente un *application server* Java Enterprise è composto da almeno i seguenti moduli:

- contenitore di componenti server-side (sia del Business Layer che del Presentation Layer)
- gestore degli accessi degli utenti e della sicurezza
- gestore degli accessi al database o in generale delle sorgenti di dati esterne
- gestore delle transazioni
- altri componenti per massimizzare le prestazioni come connection pool, load balancer, caching, ecc.

Gli *application server* più conosciuti ed utilizzati sono: GlassFish[4], JBoss[5], WebSphere[6] e WebLogic[7].

3.1.1.1 Business Layer

I componenti principali che possono essere utilizzati all'interno del Business Layer sono:

- Enterprise JavaBean[8]
- Java Persistence API[9]
- Java Message Service[10]

Per completezza è necessario specificare che sia Java Persistence API che Java Message Service, sono stati scorporati da Java Enterprise Edition in maniera che possano essere utilizzati anche in progetti Java Standard Edition. La loro importanza in applicazioni enterprise rimane comunque fondamentale in quanto posso essere utilizzati tramite particolari Enterprise JavaBean, inseriti appositamente nella specifica.

Per utilizzare gli Enterprise JavaBeans (EJB) nel *application server* è presente un apposito modulo, chiamato *EJB container*, che si occupa di gestirli e presenta almeno le seguenti funzionalità:

- Elaborazione delle transazioni
- Controllo della concorrenzialità
- Programmazione ad eventi tramite Java Message Service
- Gestione della persistenza tramite Java Persistence API
- Servizio di directory per elencare e nominare gli EJB
- Sicurezza (JCE[11] e JAAS[12])
- Invocazione di procedure remote tramite l'utilizzo di RMI-IIOP[13] o CORBA[14]

L'EJB container si occupa inoltre di gestire il ciclo di vita degli Enterprise JavaBeans e la *dependency injection*, ovvero l'inserimento a tempo di esecuzione dei riferimenti di cui il l'EJB necessita. Supponendo, per esempio, che un EJB abbia bisogno di utilizzare un altro EJB per funzionare correttamente, non risulta possibile inserire un riferimento diretto nel codice in quanto le istanziazioni non sono compito del programmatore ma del container. Questa problematica viene appunto superata tramite la dependency injection e le *java annotation*[15], in tal modo il container, a tempo di esecuzione, è in grado di inizializzare correttamente i riferimenti che sono stati "annotati" negli EJB:w .

Nelle *annotation* per riferirsi agli oggetti creati dal container viene utilizzato il Java Naming and Directory Interface (JNDI)[16]. Lo spazio dei nomi

è stato inoltre standardizzato dalla versione Java EE 6 dato che, nelle versioni precedenti, ogni *application server* utilizzava le sue convenzioni e questo poteva causare dei problemi di compatibilità.

Dopo aver esposto il funzionamento del container è ora possibile descrivere le tipologie di Enterprise JavaBeans che esistono, le principali categorie sono tre:

- Session EJB
- Entity EJB
- Message driven EJB

3.1.1.1.1 Session EJB Si occupano di gestire l'elaborazione delle informazioni sul server. Generalmente sono un'interfaccia tra i client e i servizi offerti dai componenti disponibili sul server. Ne esistono di due tipi:

stateful oggetti distribuiti che possiedono uno stato. Lo stato non è persistente, però l'accesso al bean è limitato ad un unico client. In Figura 3.2 è riportato il ciclo di vita.

stateless oggetti distribuiti senza uno stato associato, questa caratteristica permette un accesso concorrente alle funzionalità offerte dal bean. Non è garantito che il contenuto delle variabili di istanza si conservi tra diverse chiamate ai metodi del bean. In Figura 3.3 è riportato il ciclo di vita.

E' importante sottolineare che entrambe le tipologie di Session EJB possono avere due livelli di acceso per i metodi:

locale in cui i metodi dichiarati saranno accessibili solo dagli EJB che fanno parte della stessa Enterprise Application.

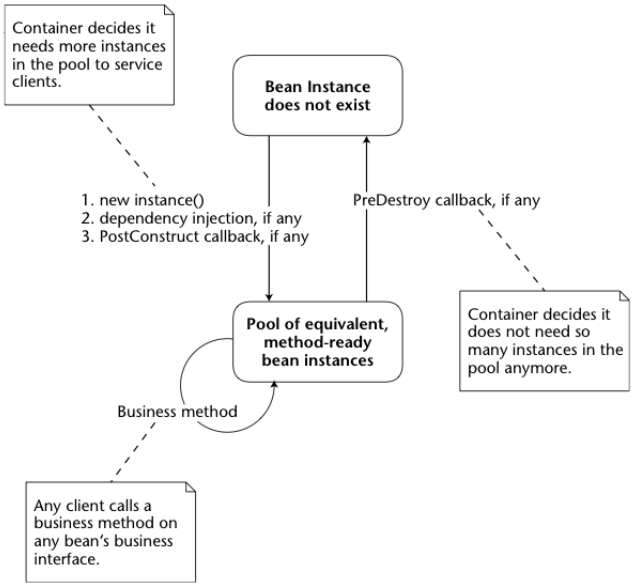

Figura 3.2: Ciclo di vita di un session bean stateful

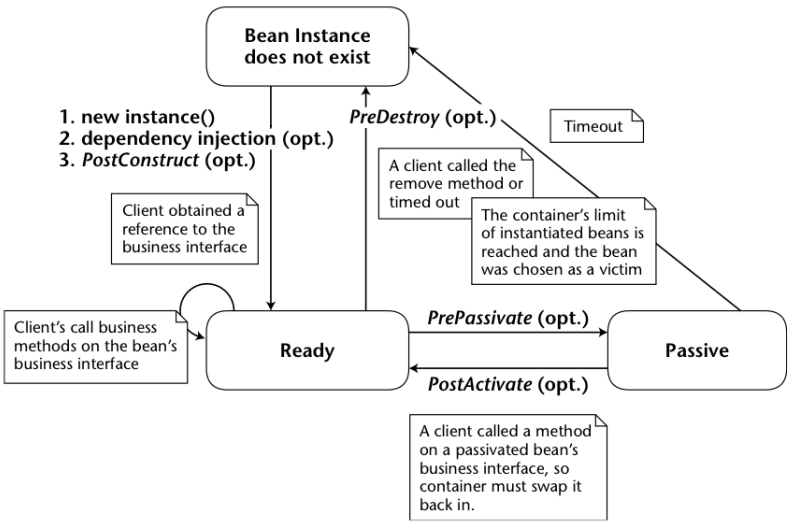

Figura 3.3: Ciclo di vita di un session bean stateless

remote in cui i metodi dichiarati saranno accessibili anche da applicazioni esterne tramite RMI-IIOP[13]. Da notare che è possibile accedere a tali metodi anche da applicazioni Java Standard Edition tramite apposite librerie.

Nella versione 6 della specifica è stata inoltre introdotta la possibilità di creare dei Session EJB che implementano il design patter *Singleton*[17], cosa che non era possibile fino alla versione precedente.

E' inoltre interessante notare che, sempre nella versione 6, è stato migliorato il supporto ai *Timer EJB*, ovvero dei Session EJB la cui esecuzione dei metodi può essere schedulata per eseguire dei compiti ad intervalli di tempo prefissati.

3.1.1.1.2 Entity EJB Gli Entity EJB permettono di gestire la persistenza dei dati tramite le Java Persistence API. Essi implementano il design pattern Object-Relational Mapping (ORM) in cui ad ogni tabella del database è associata una classe ed ad ogni tupla un'istanza di tale classe. In Figura3.4 ne è riportato un esempio.

La gestione degli Entity EJB avviene, come citato in precedenza, tramite le Java Persistence API.

Nell'utilizzo degli Entity EJB risulta importante lo stato del ciclo di vita in cui si trovano (Figura 3.5):

new appena instanziato, esiste quindi l'oggetto ma non la tupla nella tabella

managed esiste sia l'oggetto che la tupla sulla tabella

removed esiste l'oggetto ma la tupla sulla tabella è stata rimossa

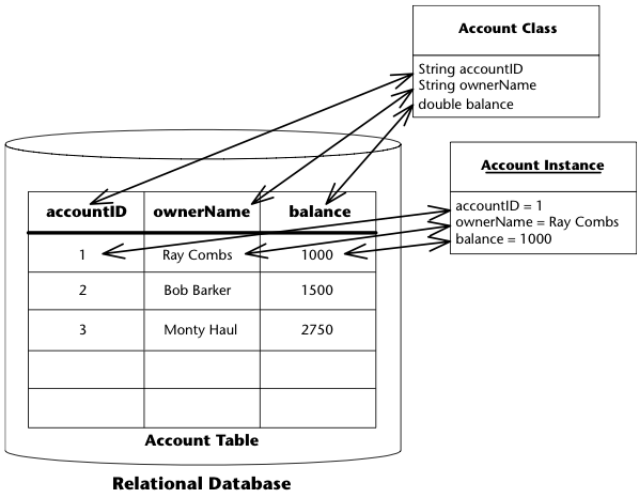

Figura 3.4: Esempio di Object-Relational Mapping

detached l'oggetto è stato spostato in un ambiente che non supporta la sua persistenza (per esempio tramite la serializzazione). Se viene rimandato indietro può essere eseguito il *merge* con il database

3.1.1.1.3 Message driven EJB I Message driven EJB sono gli unici bean con funzionamento asincrono. Quando vengono instanziati (sempre dal

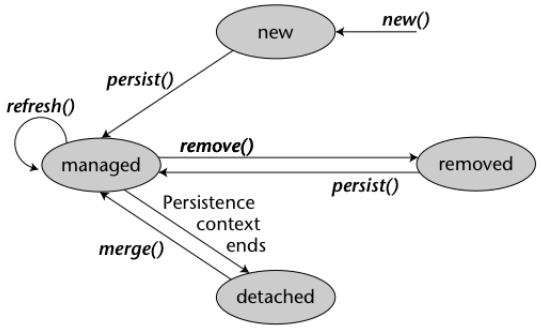

Figura 3.5: Ciclo di vita di un bean entity

container) si registrano presso un Java Message Service Server e si attivano alla ricezione di un messaggio. In Figura 3.6 è riportato il ciclo di vita.

La specifica Java Message Service (JMS), prevede due tipologie di servizi da mettere a disposizione dei Message driven EJB: le *code* e i *topic* (Figura 3.7). La coda implementa il classico paradigma *one-to-one* in cui è presente un *publisher* che invia un messaggio ad uno specifico *subscriber*. I topic invece seguono il paradigma *many-to-many* in cui uno (o più) publisher inviano un messaggio al server che si occupa di recapitarlo a tutti i subscriber che si sono registrati.

La specifica prevedere inoltre che le sottoscrizioni ai servizi possano essere *durable* o *non durable* (Figura 3.8). In caso di sottoscrizione durable, se il subscriber non è connesso, il server mantiene una copia dei messaggi che vengono inviati e provvede a consegnarli appena il subscriber torna online. Con la sottoscrizione non durable invece il subscriber riceve solo ed esclusivamente i messaggi che vengono inviati mentre è connesso al server. E' da sottolineare che utilizzare delle sottoscrizione durable implica un utilizzo notevolmente maggiore delle risorse del server JMS, in quanto le copie dei messaggi ancora da consegnare potrebbe richiedere un notevole spazio in memoria primaria o secondaria.

3.1.1.2 Presentation Layer

I componenti più conosciuti che possono essere utilizzati all'interno del Presentation Layer sono:

- Servlet[18]
- JavaServer Pages (JSP)[19]
- JavaServer Faces (JSF)[20]

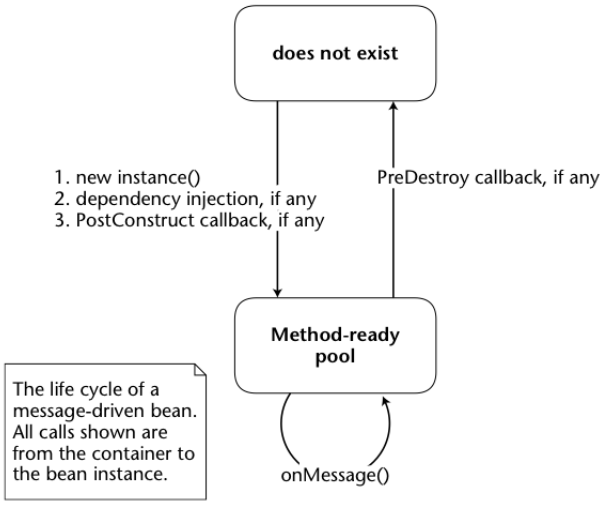

Figura 3.6: Ciclo di vita di un bean message-driven

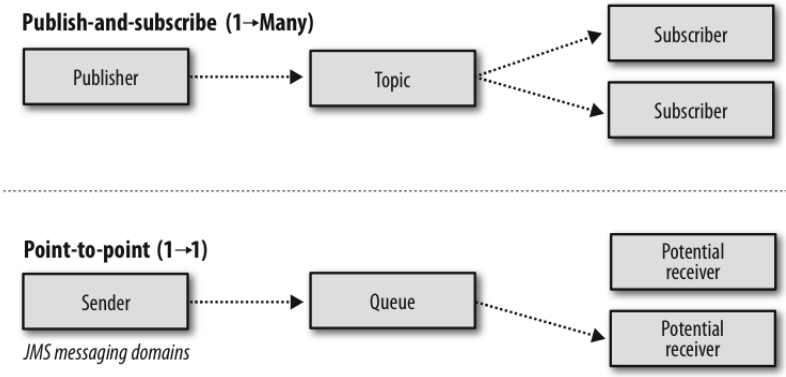

Figura 3.7: JMS Topic e Queue

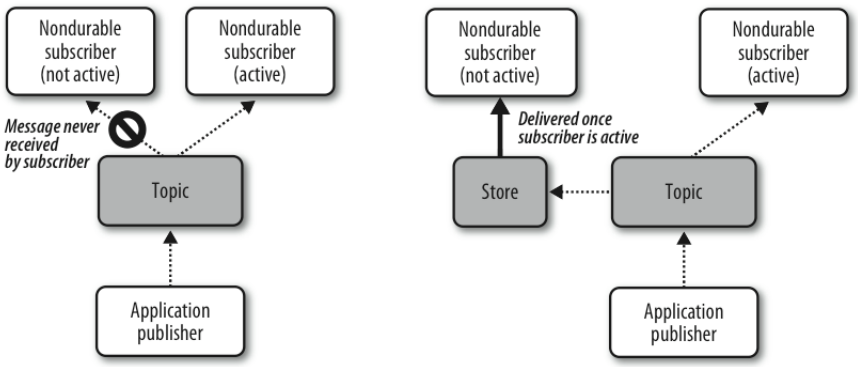

Figura 3.8: Sottoscrizione *non durable* (sinistra) e *durable* (destra)

- Google Web Toolkit (GWT)[21]

Affinché nell'*application server* sia possibile utilizzare tali componenti è presente un modulo, chiamato *Web container*, che si occupa di gestirli.

3.1.1.2.1 Servlet Scopo delle servlet è generare pagine web in forma dinamica a seconda dei parametri della richiesta inviata dal browser. Vengono però anche utilizzate spesso per la realizzazione della parte di *Controller* nel pattern di tipo *Model-View-Controller* (MVC)[17]. In tale architettura la servlet si occupa di recuperare i dati dal modello (il Business layer) e di inviarli ai componenti della vista (come JSP o JSF) in modo che siano elaborati per la visualizzazione da parte dell'utente.

3.1.1.2.2 JavaServer Pages (JSP) JavaServer Pages, di solito indicato con l'acronimo JSP, è una tecnologia Java per lo sviluppo di applicazioni Web che forniscono contenuti dinamici in formato HTML o XML. Si basa su un insieme di speciali tag con cui possono essere invocate funzioni predefinite o codice Java (JSTL). In aggiunta, permette di creare librerie di nuovi tag che estendono l'insieme dei tag standard (JSP Custom Tag Library). Le librerie

di tag JSP si possono considerare estensioni indipendenti dalla piattaforma delle funzionalità di un Web server.

Nel contesto della piattaforma Java, la tecnologia JSP è correlata con quella dei servlet. All'atto della prima invocazione, le pagine JSP vengono infatti tradotte automaticamente da un compilatore JSP in servlet. Una pagina JSP può quindi essere vista come una rappresentazione ad alto livello di un servlet.

3.1.1.2.3 JavaServer Faces (JSF) Java Server Faces è un framework che permette di realizzare *Graphical user interface* (GUI) in ambiente Enterprise, utilizzando un procedimento molto simile a quello adottato per svolgere il corrispettivo lavoro in ambito desktop. Nato dalla necessità di fornire un modello standard per la gestione delle interfacce grafiche in ambito enterprise, è stato implementato in modo da applicare sistematicamente il pattern MVC (Model-View-Controller) e introdurre un modello programmatico basato sulla gestione degli eventi. I principali vantaggi nell'uso di questo framework risultano essere:

- Esistenza di componenti predefiniti che consentono allo sviluppatore di realizzare in breve tempo interfacce web semplicemente "collegando" elementi di business logic tramite catene di eventi.

- Elementi GUI "intelligenti" in grado di validare i dati inseriti dall'utente e di archiviare e caricare *on-demand* il proprio stato da bean memorizzati lato, server denominati *model-object*.

- Indipendenza dal markup language: ogni modello d'interazione lato server viene realizzato lato client tramite *renderer* diversificati, che producono un interfaccia ottimizzata rispetto alla piattaforma utilizzata dall'utente.

3.1.1.2.4 Google Web Toolkit (GWT) Google Web Toolkit è un framework open source, sviluppato da Google, che permette di creare applicazioni Web con AJAX, scrivendo le proprie pagine esclusivamente in linguaggio Java. Sarà la libreria GWT a tradurre il codice Java e produrre le pagine HTML e JavaScript corrispondenti.

Il framework risulta estremamente innovativo in quanto permette di programmare applicazione web sfruttando gli stessi concetti che si usano per la programmazione delle GUI sui sistemi desktop. Sono quindi presenti elementi come panelli, eventi, widget, ecc. che vengono compilati ed eseguiti come codice JavaScript, dando l'impressione all'utente di utilizzare un'applicazione vera e propria e non di visualizzare una pagina sul browser.

La possibilità di scrivere il codice della presentation logic in Java "puro" presenta inoltre notevoli vantaggi:

- Possibilità di sfruttare la programmazione ad oggetti ed i Design Patterns
- Possibilità di eseguire un debug del codice molto approfondito
- Possibilità di scrivere *Unit Test* tramite JUnit[22]
- Garantire il corretto funzionamento su tutti i browser non è più compito del programmatore ma viene demandato al framework, che si occupa di gestire le varie incompatibilità in modo trasparente.

Risulta inoltre estremamente interessante la possibilità di poter facilmente integrare chiamate AJAX verso l'application server.

AJAX, acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo per la realizzazione di applicazioni web interattive (Rich Internet Application). Lo sviluppo di applicazioni HTML con AJAX si basa su

uno scambio di dati in background fra web browser e server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente. AJAX è asincrono nel senso che i dati sono richiesti al server e caricati in background senza interferire con il comportamento della pagina esistente (Figura 3.9). E' una tecnica multi-piattaforma utilizzabile su molti sistemi operativi, architetture e browser web, ed esistono numerose implementazioni open source di librerie e framework.

La tecnica Ajax utilizza una combinazione di:

- HTML (o XHTML) e CSS per il markup e lo stile
- DOM (Document Object Model) manipolato attraverso un linguaggio come JavaScript per mostrare le informazioni ed interagirvi
- XMLHttpRequest per l'interscambio asincrono dei dati tra il browser dell'utente e l'application server
- in genere viene usato XML come formato di scambio dei dati, anche se di fatto qualunque formato può essere utilizzato

AJAX risulta quindi una tecnica estremamente potente ma che può non essere banale da implementare. GWT semplifica tale compito permettendo di evitare di dover manipolare direttamente il DOM, di effettuare le chiamate in JavaScript e di dover generare i contenuti delle richieste in XML. Come mostrato in Figura 3.10 è infatti sufficiente estendere una particolare servlet (*RemoteServiceServlet*) che si occuperà di ricevere ed inviare i dati al JavaScript generato tramite GWT e di trasformarli in XML sollevando il programmatore da tale compito.

Riassumendo:

- Tutto il codice dell'applicazione viene scritto in Java

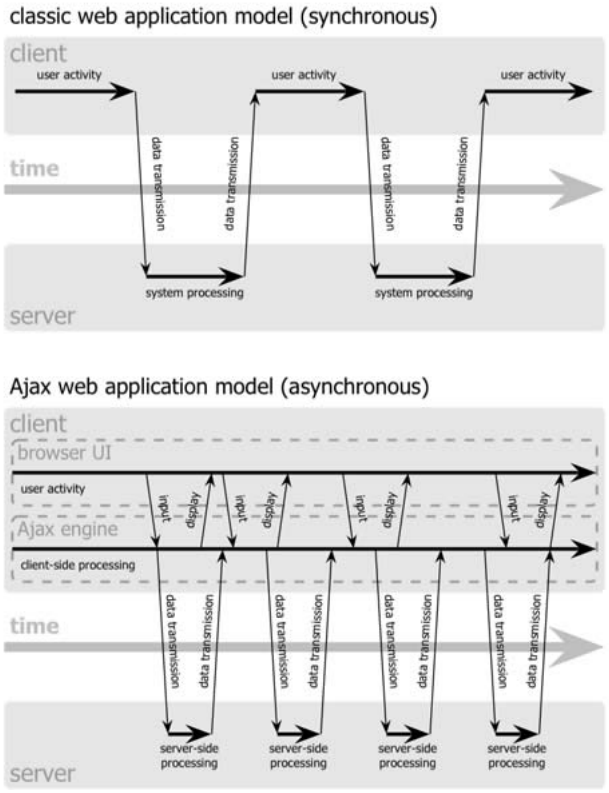

Figura 3.9: Confronto tra AJAX e il modello sincrono

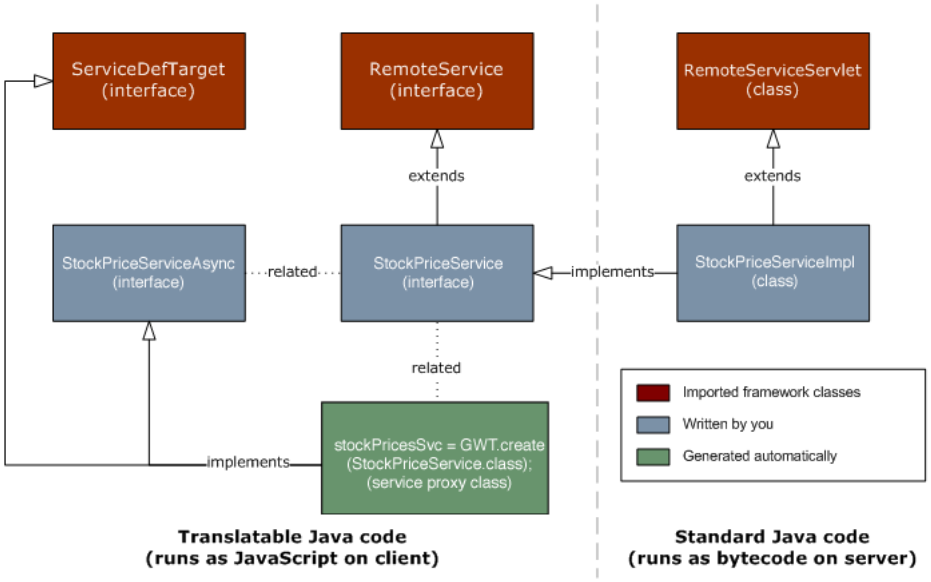

Figura 3.10: Implementazione di una chiamata AJAX con GWT

- Una parte del codice viene ricompilato in Javascript e viene eseguito nel browser dell'utente
- Le *RemoteServiceServlet* vengono caricate sull'application server e si occuperanno di gestire le richieste AJAX, interagendo con la business logic e fornendo i risultati

3.1.1.3 Clustering & Load Balancing

Durante lo sviluppo di un'applicazione enterprise è difficile prevedere in maniera precisa le risorse di cui avrà bisogno. Infatti il numero di utenti o di dati che devono essere gestiti può crescere in maniera non prevista. Se l'applicazione non viene quindi progettata pensando a questa evenienza si potrebbero presentare problemi di *performance*.

Bisogna quindi bisogna pensare l'applicazione fin dall'inizio come un *large-scale system* che viene distinto in [23] dalle seguenti caratteristiche:

- elevato numero di utenti, potenzialmente geograficamente distanti
- necessità che il sistema sia "sempre disponibile" (problematiche di *failure*, *sovraccario*, ecc.)
- necessità di poter processare un ampio numero di transazioni per secondo
- eventualità che aumenti sia il numero degli utenti che il carico di sistema
- necessità di poter essere gestito da più persone

I requisiti fondamentali per un large-scale system possono dunque essere riassunti nelle seguenti 4 proprietà:

Affidabilità Un sistema con un'affidabilità totale dovrebbe lavorare il 100% del tempo secondo le specifiche. Teoricamente quindi non ci dovrebbero essere errori nelle sue componenti che possano causare una riduzione delle *performance* o delle funzionalità. Ovviamente questo risultato non è raggiungibile in pratica, quindi un sistema verrà considerato *affidabile* quando si comporterà coerentemente anche in presenza di errori.

Disponibilità misura la percentuale di tempo in cui il sistema è disponibile per gli utenti. Un sistema può non essere disponibile per varie ragioni, come problemi sulla rete, operazioni di manutenzione, sovraccarichi, ecc.

Manutenibilità il principio della manutenibilità si basa sul fatto che più application server sono più difficili da amministrare rispetto ad uno solo. Ovviamente questa caratteristica va in contrasto con la disponibilità

Scalabilità il principio della scalabilità è quello di poter aggiungere risorse in caso di necessità. Se il carico di sistema diventa troppo elevato è

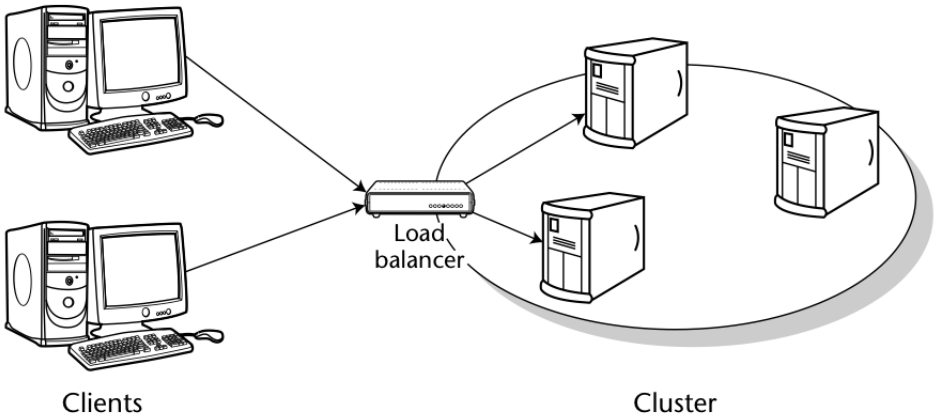

Figura 3.11: Cluster

infatti sempre possibile cambiare delle componenti per aumentare le performance ma, ad un certo punto, questa operazione può non risultare più sufficiente. In tal caso sarebbe più efficiente aggiungere un secondo server (questa operazione prende il nome di *clustering*), se però l'applicazione non è stata studiata esplicitamente per permetterlo ciò potrebbe non essere possibile.

Una delle migliori soluzioni per ottenere le caratteristiche richieste ad un large-scale system risulta appunto il *clustering*. Possiamo definire un *cluster* come un gruppo di server che fornisce un servizio ai client come se fosse uno solo. Infatti i client che utilizzano tale servizio sono solitamente all'oscuro che le loro richieste sono gestite da più server e tipicamente non hanno possibilità di scelta su quale server si occuperà delle loro richiesta (Figura 3.11). Le richieste sono infatti redirette in maniera trasparente verso un nodo del cluster per essere gestite. I client vedono dunque il cluster come un server singolo e non come un gruppo di server collaboranti tra di loro. Risulta però importante notare come il clustering sia una tecnologia altamente complessa, sono infatti necessari protocolli di replicazione, protocolli per transazioni

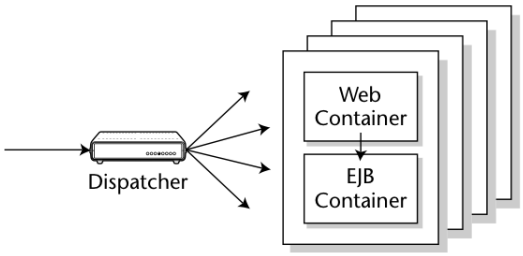

Figura 3.12: Presentation Layer unito al Business Layer

distribuite ed eventualmente componenti come *load balancer* e *traffic redirector*. Il clustering viene comunque sopportato dalla maggior parte degli application server Java EE, sia commerciali che open source, ma in maniera altamente *vendor-dependent* in quanto, al momento, non fa parte della specifica.

Come spiegato in precedenza gli application server Java EE contengono due moduli: il Web container e l'EJB container. Questo implica che è possibile utilizzare due diverse tipologie di clusterizzazione:

collocated architecture i componenti web (servlets, JSP, ecc) e quelli della business logic (EJB) risiedono sugli stessi server (Figura 3.12)

distributed architecture i componenti web e quelli della business logic risiedono su gruppi di server differenti (Figure 3.13)

Un'architettura distribuita è inevitabilmente più complessa da implementare e da gestire ed introduce ulteriori problematiche come un aumento della latenza, risulta però sicuramente più flessibile e scalabile. Il raffronto tra le due architetture è riportato nella tabella 3.2.

Caratteristica	Collocated	Distributed
Affidabilità	ALTA: non ci sono comunicazioni remote. Tutto si svolge nello stesso processo quindi sono presenti meno fattori che possono causare comportamenti imprevisti	BASSA: la presenza di più server coinvolti nella stessa richiesta aumenta i fattori che possono causare comportamenti imprevisti
Disponibilità	ALTA: se un server fallisce gli altri continuano a lavorare. Per bloccare il servizio devono fallire tutti i server	ALTA ma minore della Collocated: in caso fallisca un server gli altri continuano a lavorare, ma per bloccare il servizio è sufficiente che fallisca uno dei due gruppi di cluster
Manutenibilità	ALTA: tutti i server sono uguali quindi è più semplice da gestire	MEDIA: essendoci due tipologie di server la manutenzione è leggermente più complicata
Scalabilità	BASSA: è possibile aggiungere nuovi server ma non posso bilanciare il numero in caso la business logic abbia un carico diverso rispetto alla presentation logic	ALTA: è possibile dimensionare i gruppi secondo il carico di ogni layer

Tabella 3.2: Confronto tra architettura collocata e distribuita

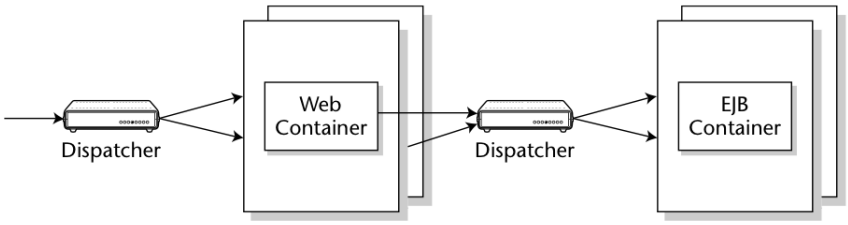

Figura 3.13: Presentation Layer separato dal Business Layer

3.1.2 SEAM

Seam[24] è un *Application Framework* che nasce con l'intento di integrare due delle principali componenti della specifica JavaEE: EJB e JSF[20].

JSF svolge la funzione di *presentation framework* e viene collocato nel *web tier* fornendo sia l'interfaccia utente (UI) che il modello per gli eventi lato server.

EJB è invece il modello di programmazione standard per creare componenti sicuri e scalabili della logica di *business*; tramite gli EJB è inoltre possibile utilizzare Java Peristence API (JPA) che definiscono un modello standard per la gestione della persistenza, in particolare occupandosi del *mapping* tra il modello relazione e le classi java.

3.1.2.1 Vantaggi

Le due tecnologie appena citate sono parte della specifica JEE e possono essere quindi liberamente utilizzate su qualsiasi *application server* che rispetta tali specifica, è da notare inoltre che per utilizzarle non è obbligatorio utilizzare nessun framework aggiuntivo come *seam*. L'obiettivo di Seam risulta dunque quello di fare da collante tra queste due specifiche in modo da rendere più facile integrarle tra loro, dato che può essere un operazione non sempre

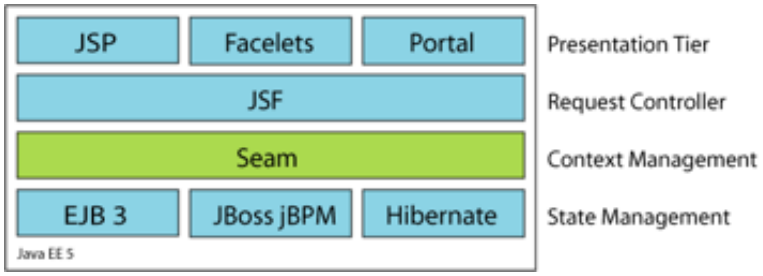

Figura 3.14: Architettura di SEAM

banale. Infatti, come mostrato in figura 3.14, SEAM si pone tra lo strato JSF ed EJB in maniera da facilitarne la comunicazione.

SEAM introduce anche un concetto non presente nella specifica JEE: il *Conversation context* [25]. Nella specifica JEE, per la parte di presentazione, sono infatti presenti i seguenti contesti (ovvero spazi logici in cui è possibile associare oggetti):

page: permette di associare oggetti ad una istanza di una pagina.

request: permette di associare oggetti ad una richiesta (tipicamente HTTP).

session: permette di associare oggetti ad una sessione associata ad ogni utente, di solito viene mantenuto lo stato tramite *cookies* o l'*url rewriting*.

application: permette di associare oggetti all'applicazione in maniera che siano accessibili da ogni componente.

Questi contesti sarebbero più che sufficienti se non si stessero sempre più affermando situazioni in cui può verificarsi un accesso concorrente alla stessa risorsa, si pensi ad esempio al caso in cui l'utente apra più *tab* del *browser* sulla stessa pagina o si utilizzino tecnologie come *AJAX* (Asynchronous JavaScript and XML). In tal caso è dunque necessario utilizzare dei meccanismi di sincronizzazione per gli oggetti che vengono condivisi (per esempio

nella sessione) al fine di mantenerli in uno stato consistente. L'utilizzo di meccanismi di sincronizzazione però può non essere sufficiente, in quanto garantisce la consistenza, ma non può assicurare che il flusso di informazioni segua il desiderio dell'utente. Prendiamo l'esempio di due *tab* aperti sulla stessa pagina di prenotazione di un hotel, l'utente vorrebbe che le due pagine (anche se si riferiscono allo stesso *url*) seguano due percorsi di elaborazione completamente separati, per esempio per verificare i costi di due camere di categoria diversa. Questo risultato può essere difficile da ottenere utilizzando esclusivamente la specifica JEE e per questo gli sviluppatori di SEAM hanno aggiunto il *Conversation context*, che permette di gestire queste situazioni garantendo la coerenza dei dati.

3.1.2.2 Limiti

Nel precedente paragrafo sono stati illustrati quali sono i vantaggi di adottare SEAM per lo sviluppo di un'applicazione *Enterprise*, bisogna però anche tenere conto delle limitazioni che tale scelta comporta.

Innanzi tutto è necessario specificare che SEAM integra le principali specifiche JEE ma ne modifica leggermente l'utilizzo, aggiungendo delle funzionalità non presenti. Se questo può portare dei vantaggi di contro rende impossibile il passaggio o l'integrazione di altri framework nel sistema. Si perde quindi la possibilità di diversificare o sostituire le scelte fatte per ogni livello dell'applicazione (presentazione, logica di business). Dato che lo sviluppo a componenti è un punto di forza della specifica JEE questa limitazione può risultare particolarmente onerosa, portando fino alle necessità della riscrittura totale dell'applicazione in caso sia indispensabile cambiarne anche solo una parte.

Un'ulteriore limitazione risulta essere la separazione tra il *web layer* e il

business layer: per come è strutturato il framework bisogna infatti effettuare il *deploy* dei due layer sullo stesso application server. La separazione è possibile ma è necessario eseguire una procedura estremamente complicata e non ben documentata. In caso si volesse quindi applicare un'architettura distribuita con la separazione dei due layer su server differenti si potrebbero riscontrare dei notevoli problemi.

3.1.2.3 Clustering

Come specificato precedentemente la separazione dei due layer può non essere banale, risulta però possibile effettuare il clustering ma solo ed esclusivamente a livello di *application server* (collocated). Seam viene infatti sviluppato a stretto contatto con il team che si occupa di sviluppare JBoss[5], un application server che supporta il clustering a partire dalla versione 3.0 e la garantisce per Seam.

Per supporto al clustering si intende, come è stato spiegato precedentemente, che le varie istanze in esecuzione (che siano su macchine diverse o sulla stessa macchina) sono in grado di colloquiare "dietro le quinte" dell'applicazione enterprise per scambiarsi informazioni riguardanti lo stato della stessa. Ciò è necessario perché il cluster sia visto nel suo complesso come un unico elaboratore, quindi eventuali modifiche allo stato dell'applicazione che avvengono in un'istanza devono essere trasmesse (o condivise) su tutti i nodi del cluster. Un esempio può essere lo stato dei Session Bean Stateful, degli Entity Bean, o della cache. Queste funzionalità vengono gestite dal motore di clustering dell'application server in coppia ad un dispositivo di bilanciamento del carico e risultano trasparenti ai programmatori.

3.1.3 Spring

Spring è un framework open source per lo sviluppo di applicazioni su piattaforma Java.

La prima versione venne scritta da Rod Johnson e rilasciata insieme con la pubblicazione del proprio libro[26].

All'inizio il framework venne rilasciato sotto Licenza Apache nel giugno 2003. Il primo rilascio importante è stato il primo del marzo 2004, seguito da due successivi rilasci importanti nel settembre 2004 e nel marzo 2005.

Spring è stato largamente riconosciuto all'interno della comunità Java quale valida alternativa al modello basato su Enterprise JavaBeans (EJB). Rispetto a quest'ultimo, il framework Spring lascia una maggiore libertà al programmatore fornendo allo stesso tempo un'ampia gamma di soluzioni semplici adatte alle problematiche più comuni.

Sebbene le peculiarità basilari di Spring possano essere adottate in qualsiasi applicazione Java, esistono numerose estensioni per la costruzione di applicazioni web-based costruite sul modello della piattaforma JavaEE. Questo ha permesso a Spring di raccogliere numerosi consensi e di essere riconosciuto anche da importanti vendor commerciali quale framework d'importanza strategica.

Spring è famoso per la sua architettura modulare (Figura 3.15) e si basa su due concetti fondamentali:

- Aspect-oriented programming
- Inversion of Control

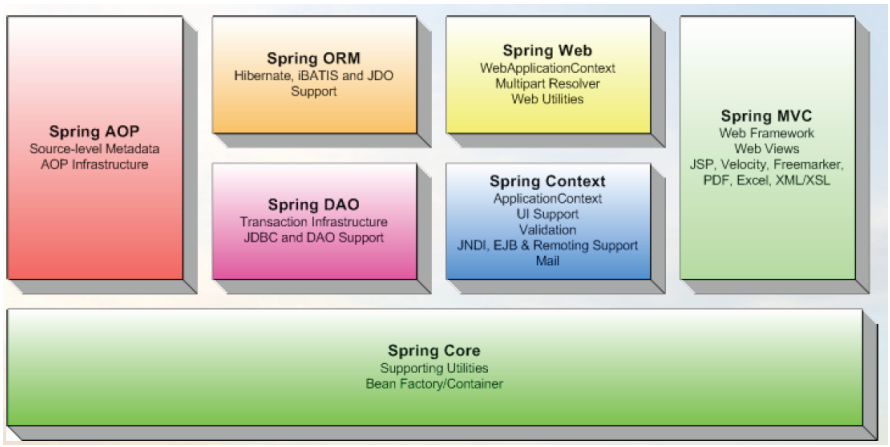

Figura 3.15: Architettura di Spring

3.1.3.1 Aspect-oriented programming (AOP)

La *programmazione orientata agli aspetti* è un paradigma di programmazione basato sulla creazione di entità software, denominate aspetti, che sovrintendono alle interazioni fra oggetti finalizzate ad eseguire un compito comune. Il vantaggio rispetto alla tradizionale programmazione orientata agli oggetti consiste nel non dover implementare separatamente in ciascun oggetto il codice necessario ad eseguire questo compito comune. Un esempio possono essere le funzioni di controllo degli accessi, la verifica se si è in possesso dei permessi necessari ad eseguire una certa operazione non dovrebbe essere inserita nel codice dell'oggetto, in quanto non ha nessuna relazione con le operazioni logiche che il metodo deve eseguire ed inoltre risulta più conveniente che tale funzione venga condivisa anche da altri oggetti. Una funzione estremamente importante che Spring mette a disposizione con la tecnica AOP è la gestione delle transazioni che in questa maniera può essere anche modificata al momento del deploy, dato che viene impostata in un file di configurazione.

In sintesi un programma aspect-oriented è costituito essenzialmente da due insiemi di costrutti: gli aspetti e gli oggetti. Gli aspetti sono delle entità

esterne agli oggetti che osservano il flusso del programma generato dalle interazioni tra oggetti, modificandolo quando opportuno.

3.1.3.2 Inversion of Control (IoC)

Inversion of Control è un pattern di programmazione secondo il quale si deve tendere a tener disaccoppiati i singoli componenti di un sistema. Le eventuali dipendenze non vengono scritte all'interno del componente stesso, ma vengono iniettate dall'esterno: non si segue il normale flusso di controllo e gli oggetti quindi non instanziano e richiamano gli oggetti dal quale il loro lavoro dipende, ma queste funzionalità vengono fornite da un ambiente esterno tramite dei contratti definiti da entrambe le entità (tramite delle interfacce).

La Dependency Injection è una delle tecniche con le quali si può attuare l'IoC. Essa prende il controllo su tutti gli aspetti di creazione degli oggetti e delle loro dipendenze. Spring usa molto diffusamente la Dependency Injection con il risultato, tra le altre cose, di eliminare dal codice applicativo ogni logica d'inizializzazione. Normalmente, senza l'utilizzo di questa tecnica, se un oggetto richiede di accedere ad un particolare servizio, l'oggetto stesso si prende la responsabilità di gestirlo, o avendo un diretto riferimento al servizio, o individuandolo con un *Service Locator* che gli restituisce un riferimento ad una specifica implementazione del servizio. Con l'utilizzo della dependency injection un riferimento ad una implementazione di questo servizio gli viene iniettata dal framework esterno, senza che il programmatore che crea l'oggetto sappia nulla sul suo posizionamento del servizio o altri dettagli sullo stesso.

Un altro punto interessante dell'IoC è la semplificazione degli *Unit Test*. Infatti per i test di ogni oggetto si può facilmente creare delle implementazioni

di comodo per le entità dipendenti ed effettuare il test di unità per la sua logica, in maniera esplicita e in isolamento. Questi test sono più facili da scrivere e più veloci. La maggior parte dei test di unità infatti non richiederà di creare implementazioni fasulle di connessioni a database e alberi JNDI, il che può dare origine a test poco performanti.

3.1.3.3 Clustering

Purtroppo, dalla documentazione reperita, non risulta che sia facilmente eseguibile il *clustering* di un'applicazione Spring. Questo risulta essere un notevole limite che può essere eventualmente superato utilizzando *Terracotta* [27].

Terracotta è un software open-source per effettuare clustering al livello della Java Virtual Machine. Con esso è teoricamente possibile trasformare un'applicazione *multithred* scritta per essere eseguita su una singola JVM in un'applicazione distribuita. Risulta comunque essere una tecnologia estremamente innovativa ed non è stata trovata letteratura riguardate il suo utilizzo in ambienti *mission critical*.

3.2 Ruby

Ruby è un linguaggio di scripting completamente a oggetti. Nato nel 1993 ad opera del giapponese Yukihiro Matsumoto. Ruby è stato il primo linguaggio di programmazione sviluppato in Oriente a guadagnare abbastanza popolarità da superare la barriera linguistica che separa l'informatica nipponica da quella internazionale e ad essere usato anche in Occidente in progetti di rilievo.

Ruby, come Java viene eseguito su una VM (sono disponibili diverse implementazioni del linguaggio). Ruby a differenza di Java è un linguag-

gio dinamicamente tipizzato[1] E' stato accettato come standard industriale Giapponese nel 2011 (JIS X 3017) e dal 2012 è stato anche accettato come standard internazionale (ISO/IEC 30170). Inizialmente Ruby non ha goduto di una notevole popolarità per mancanza di una documentazione in lingua inglese, documentazione fornita solo nell'anno 2000.

Negli ultimi anni la popolarità di Ruby ha subito una forte impennata, dovuta alla comparsa di framework di successo per lo sviluppo di applicazioni web, come Nitro e Ruby On Rails, nonché del Metasploit Framework, ambiente per la creazione e l'esecuzione facilitata di exploit. Il livello di maturità raggiunto dal linguaggio, l'enorme ecosistema creatosi attorno ad esso e la produttività di esso hanno fatto in modo che questo sia utilizzato anche in ambito enterprise, in progetti di grosse dimensioni dove è richiesto un certo livello di servizio.

3.2.1 Ruby On Rails

Ruby on Rails è un framework per lo sviluppo di applicazioni web scritto in Ruby da David Heinemeier Hansson come progetto interno presso 37signals[2]. L'architettura del framework è fortemente ispirata al paradigma Model-View-Controller (MVC) e fa un forte utilizzo di due principi:

- *DRY Don't Repeat Yourself*[3]
- *Convention Over Configuration*[4]

[1] http://it.wikipedia.org/wiki/Tipizzazione_dinamica

[2] http://37signals.com/

[3] Noto anche come Single Point of Truth è un principio di progettazione e sviluppo secondo cui andrebbe evitata ogni forma di ripetizione e ridondanza logica nell'implementazione di un sistema software.

[4] è un paradigma di programmazione che prevede configurazione minima (o addirittura

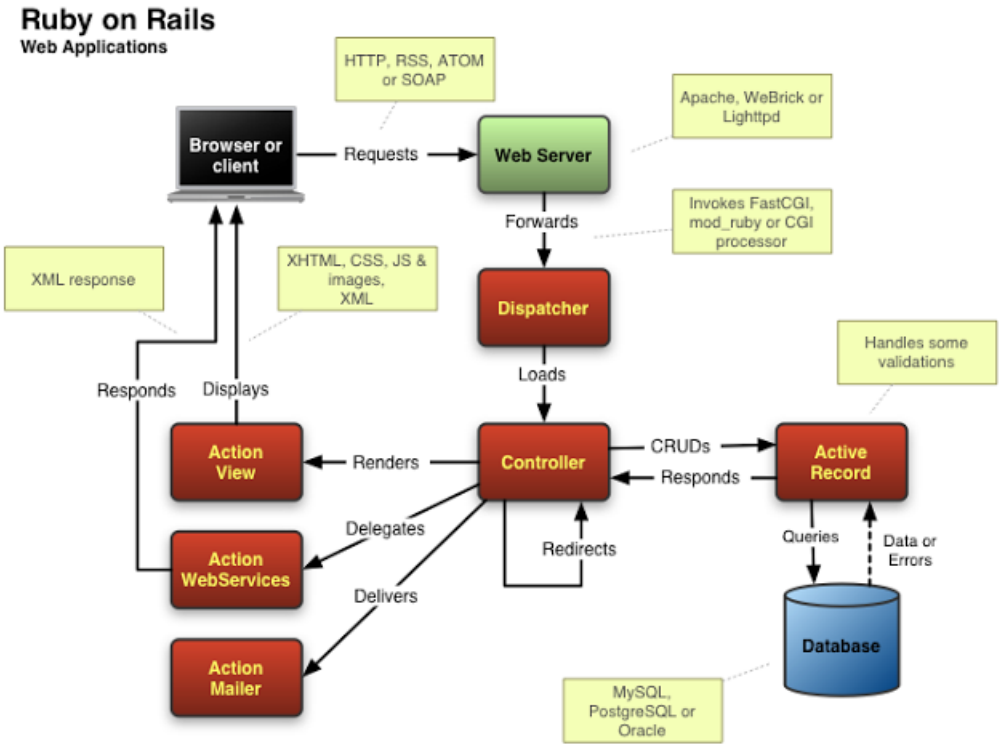

Figura 3.16: Architettura di Ruby On Rails

Ruby On Rails con l'avvento della versione 3.2 e adesso della versione 4, unitamente alla versione 2.0 del linguaggio Ruby ha raggiunto un ottimo livello di maturità e garantisce prestazioni che si avvicinano a quelle del linguaggio Java. È in ogni caso presente un'implementazione del linguaggio scritta in linguaggio Java, chiamata JRuby. Questa implementazione esegue una compilazione *just-in-time* o *ahead-of-time* in un bytecode Java. JRuby inoltre fa utilizzo di altre feature della JVM come il garbage collector che presenta prestazioni elevate.

Rails ha un architettura totalmente modulare (Figura 3.16) e adotta il design pattern architetturale Model-View-Controller (MVC)[5]. Ruby on Rails include un ORM, *Active Record*, che mappa tabelle del database in classi Ruby, record delle tabelle in oggetti e colonne in attributi degli oggetti. Anche questo modulo del framework è totalmente indipendente (può essere utilizzato anche senza framework) e può essere sostituito con un altro modo che adempie allo stesso compito (per esempio *Data Mapper*). Active Record è totalmente database agnostic e sono presenti driver per tutti i più conosciuti RDBMS, sia commerciali che open source. La modularità di Ruby on Rails permette inoltre di utilizzare il framework anche con database di tipo NoSQL (MongoDB, CouchDB, Redis, ecc.).

Per quanto riguarda il presentation layer, Ruby on Rails permette di integrare facilmente qualsiasi javascript framework (BackboneJS, AngularJS, EmberJS, ecc.), per offrire una maggiore esperienza utente e creare applicazioni che spostano parte della parte di visualizzazione dal lato server al lato

assente) per il programmatore che utilizza un framework che lo rispetti, obbligandolo a configurare solo gli aspetti che si differenziano dalle implementazioni standard o che non rispettano particolari convenzioni di denominazione o simili.

[5]alcuni esperti tendono a sottolineare che probabilmente il design architetturale che si avvicina di più a quello di Ruby on Rails sia *Model2*: http://en.wikipedia.org/wiki/Model2

client, creando un livello di interattività nettamente maggiore.

Ruby on Rails fornisce tutti gli strumenti e l'integrazione con tutti i tool per permettere allo sviluppatore di seguire un processo di sviluppo *Agile*[6]. Fornisce il supporto per Test Driven Development (TDD) e Behavior Driven Development (BDD), compilazione delle assets (immagini, fogli di stile e script javascript) per poter essere eventualmente servizi da Content Delivery Network (CDN) e per permettere delle performance migliori dell'applicazione in produzione. Rails offre una struttura di directory che permettono una precisa organizzazione del codice e induce il programmatore ad eseguire una corretta *separation of concerns* e ad adottare il principio di *Don't Repeat Yourself*. Nativamente presenta diversi environment (development, test e production, ma possono esserne creati altri) per offrire feature differenti a seconda delle esigenze.

Per quanto riguarda il deploy di applicazioni risulta essere presente un ottimo tool, *Capistrano* che ne permette la totale automatizzazione e fornisce anche meccanismi di fallback in caso di fallimento. Per quanto riguarda l'esecuzione di applicazioni Ruby on Rails ci sono diversi application server che possono essere utilizzati:

- Mongrel
- Phusion Passenger (Apache mod_rails)
- Unicorn
- Thin
- Puma
- Torquebox (JRuby)

[6]http://agilemanifesto.org/

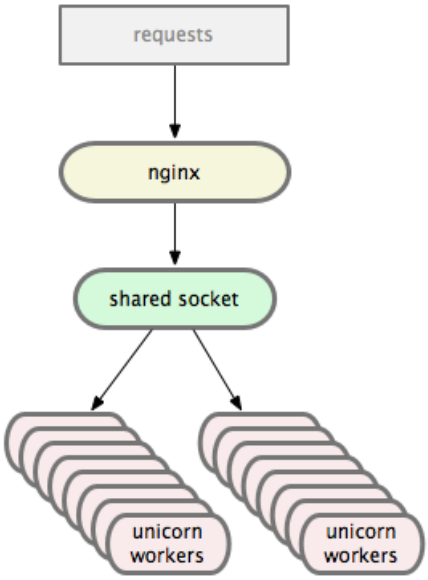

Figura 3.17: Unicorn worker pool

- Glassfish (JRuby)
- Jetty (JRuby)

La lista non è completa e sono presenti un grande numero di prodotti minori.

La configurazione varia in base alle esigenze. Per singola applicazione web con basso carico si può utilizzare direttamente Phusion Passenger, totalmente basato su Apache. Per situazioni che richiedono maggiori prestazioni e un certo livello di servizio sarà efficacie avere configurazioni più complesse ma che permettono performance migliori e livelli di servizio più elevati. Una possibile soluzione, con ottime performance e notevoli vantaggi è quella illustrata nelle Figure 3.17 e 3.18.

La configurazione di Figura 3.17 prevede un server *nginx*[7] che agisce da *Reverse Proxy* e spedisce richieste attraverso un *Unix Domain Socket* (o TCP

[7]http://nginx.org/

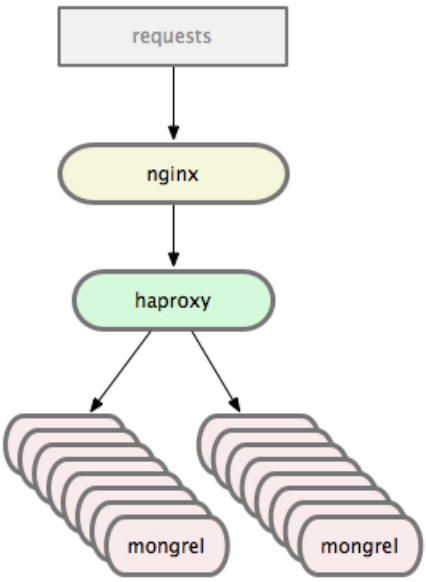

Figura 3.18: Mongrel cluster

se si preferisce, ma si avranno performance inferiori). Sarà presente un processo Unicorn master (non prenderà carico di nessuna richiesta in ingresso) che gestirà i vari workers mentre il bilanciamento sarà affidato al sistema operativo. Il processo di bootstrap avviene nel seguente modo: quando il processo Unicorn master inizia, carica in memoria l'applicazione. Come esso è pronto per servire le richieste esegue il fork di 16 workers. Questi workers eseguiranno una *select()* sul socket e serviranno solo le richieste che sono capaci di gestire. In questo modo il kernel gestirà il bilanciamento del carico. In caso di workers che impiegano più di un certo numero di secondi a completare l'operazione il master provvederà a eseguire un kill su tale worker e a creare un fork di un nuovo. Il nuovo worker servirà immediatamente una nuova richiesta. In caso di un uso eccessivo di memoria da parte di un worker questo potrà essere ucciso e creato un nuovo worker attraverso l'operazione di fork. Con questo tipo di configurazione si implementa inoltre uno *Zero-*

downtime deployments, dove non si avranno momenti in cui il servizio risulta offline per un deployment in corso. Inoltre i workers lavorano in ottica pull, ovvero saranno essi a richiedere una richiesta nel momento in cui potranno gestirla

Invece la configurazione di Figura 3.18 risulta sostanzialmente differente per il tipo di application server utilizzato e per altre caratteristiche tipiche di esso.

Come possiamo vedere dalla figura abbiamo un server nginx che agisce sempre da reverse proxy ad un pool di processi Mongrel utilizzando un bilanciatore o un semplice algoritmo *round robin*. Eventualmente se è necessaria una maggiore affidabilità è possibile utilizzare un *HAProxy*[8].

[8]http://haproxy.1wt.eu/

Capitolo 4

Soluzione proposta: TCO Web Application

Nei capitoli precedenti è stato introdotto il problema che il lavoro di tesi si propone di risolvere e le possibile tecnologie che possono essere utilizzate.

In questo capitolo si illustreranno le tecnologie che sono state ritenute più appropriate e l'architettura del sistema che è stata scelta.

Riepiloghiamo i compiti principali a cui l'applicazione deve adempiere:

- Garantire l'integrità dei dati.
- Prevedere meccanismi di fault tolerance e fornire risposte consistenti anche in situazioni di guasto.
- Garantire un livello di disponibilità elevato.
- Essere facilmente estendibile.
- Integrazione nell'ambiente enterprise.
- Interazione con sistemi esistenti.

- Permettere una capillare gestione degli accessi.
- Fornire un alto livello di usabilità dell'applicazione.

Dall'analisi svolta sulle varie tecnologie, ne sono state scelte due e si è deciso di procedere alla creazione di due prototipi minimali per fare un ulteriore confronto.

4.1 Realizzazione dei prototipi

Le tecnologie scelte per lo sviluppo dei due prototipi sono state:

- Ruby On Rails
- GWT

Ruby On Rails è stato scelto perchè è risultato un framework con una altissima produttività e che forniva un ecosistema di librerie molto vasto che permetteva di adempiere ai compiti sopra descritti, come per esempio l'integrazione nell'ambiente enterprise per la gestione degli accessi. L'unico inconveniente incontrato nello sviluppo di questo prototipo è stato nel mappare le classi Ruby con le tabelle del database. Il database consiste in circa 80 tabelle, molte di esse presentano chiavi primarie composte, questo è stato inizialmente problematico poichè l'ORM di Ruby On Rails, Active Record, per convenzione utilizza una colonna id come chiave primaria. Sono possibili differenti configurazioni, ma non è previsto nativamente il supporto per chiavi primarie composte. Dopo una attenta ricerca è stata trovata una gemma che implementa questa funzionalità ed estende il funzionamento di Active Record per poter agire su tabelle con chiavi primarie composte. Per quanto riguarda il presentation layer sono state create viste con tecnologie standard:

Haml, Css e una minima parte di Javascript. In questa fase si è evitato di introdurre un Javascript framework per il presentation layer poichè non era ancora delineata una chiara scelta.

Il secondo prototipo è stato sviluppato con la tecnologia GWT, poichè era di interesse capire come questa scelta potesse gestire il presentation layer. Per quanto riguarda il business layer abbiamo adottato una soluzione software che simulasse tale componente e che fornisse staticamente alcuni dati per popolare il prototipo. Questa scelta è stata fatta perchè già a conoscenza di non incontrare nessuna problematica nel mapping tra EJB Entity e tabelle con chiavi primarie composte. Questo inoltre ha permesso di risparmiare una notevole quantità di tempo nella creazione di questo prototipo.

4.2 Tecnologie scelte

Dopo avere analizzato attentamente i due prototipi e le soluzioni tecnologiche scelte fino ad adesso si è deciso di utilizzare la specifica Java EE 6 con GWT come tecnologia per il presentation layer. I due prototipi hanno presentato ottime caratteristiche e nessuno dei due ha fallito nei requisiti specifici che l'applicazione si proponeva. Per questo siamo andati ad analizzare più in profondità le esigenze dell'azienda stessa relative a tale scelta e sono comparsi, in seconda analisi, due requisiti non espressi esplicitamente prima:

- Facilità nel reperire personale capace di manutenere l'applicazione.
- Minimizzare l'utilizzo di altre tecnologie di contorno per avere un solo dominio tecnologico.

L'azienda possiede un enorme know-how interno, ma principalmente focalizzato su tecnologie enterprise già affermate. Dopo un analisi delle risorse

interne dell'azienda la scelta di Java EE 6 è risultata chiara. Inoltre l'utilizzo di GWT ha permesso di minimizzare altre conoscenze di contorno, come quella di Javascript nativo, HTML e CSS, poichè anche tutte le viste dell'applicazione saranno sviluppate in linguaggio Java (e anche tool visuali) e saranno poi automaticamente compilate in Javascript nativo in fase di compilazione dell'applicazione.

Appurato l'utilizzo della specifica Java EE v6 come soluzione più appropriata si è poi quindi passati alla scelta delle tecnologie correlate con essa, per quanto riguarda il Business Layer ed il Presentation Layer.

Dopo un analisi sono state effettuate le seguenti scelte:

Business Layer: Enterprise Java Bean

Presentation Layer: Google Web Toolkit

A seguire verranno specificati i vantaggi di tale scelta.

4.2.1 Business Layer: Enterprise Java Bean

Gli EJB risultano essere una scelta obbligata per il business layer se si decide di utilizzare Java EE, in quanto ne rappresentano il fondamento. Si può inoltre affermare che sono le caratteristiche di modularità e scalabilità degli EJB che hanno portato a considerare Java EE la scelta più adatta a questo progetto. Si è deciso di utilizzare l'ultima *release* della specifica Jave EE, la 3.1, rilasciata a fine 2009[28]. Tale scelta è risultata non vincolata a particolari scelte dell'application server poichè vi era totale libertà in questa direzione.

4.2.2 Presentation Layer: Google Web Toolkit

Mentre per il Business Layer la scelta era in sostanza obbligata, per il Presentation Layer è presente una più ampia gamma di tecnologie che possono essere utilizzate. Come citato nel precedente capitolo le opzioni più comuni possono essere:

- JavaServer Pages (JSP)
- JavaServer Faces (JSF)
- Google Web Toolkit (GWT)

Alla fine la scelta è ricaduta su GWT, in quanto anche per il dominio logico dell'applicazione da creare, avere un *look and feel* simile a quello di un applicazione desktop (ma eseguita su browser) potesse essere una scelta valida da un punto di vista di UX. Inoltre la sola conoscenza del toolkit e del linguaggio Java (e minime conoscenze di HTML/CSS/Javascript). GWT risultata inoltre una tecnologia matura abbastanza da utilizzare in questo ambito, infatti come descritto in[29], le web application di Google (Maps, Document, ecc.) sono state tutte sviluppate utilizzando questa tecnologia.

JavaServer Faces e Google Web Toolkit sono invece stati appositamente sviluppati per supportare nativamente AJAX. La scelta tra i due si è quindi basata su altri fattori. Nello specifico la possibilità, offerta da GWT, di sviluppare web application direttamente in linguaggio Java lo ha fatto prevalere sulle JSF. Risultano infatti evidenti i seguenti vantaggi:

- Possibilità di sfruttare la programmazione ad oggetti ed i Design Patterns
- Possibilità di eseguire un debug del codice lato client

- Possibilità di scrivere *Unit Test*

La nostra applicazione si proponeva inoltre di avere una buona UX e una chiara UI. Per adempiere a questi compiti su web è spesso necessario ricorrere anche ad AJAX per garantire un applicazione più interattiva e che guidi l'utente nel suo utilizzo. Adempiere a questi compiti soltanto con normale HTML e CSS e utilizzo di Javascript in maniera sincrona non risulta possibile. GWT permette di raggiungere questo obiettivo in maniera semplice e utilizzando solo il linguaggio Java, senza aver l'obbligo di integrare altre tecnologie come jQuery o addirittura interi Javascript frameworks come BackboneJS, AngularJS o altri.

4.3 Architettura del sistema

Riportata in Figura 4.1 possiamo vedere come si presenterà in ottica generale la soluzione proposta.

L'idea dell'architettura è di avere una applicazione distribuita a 3 livelli e con la gestione dell'autenticazione affidata ad un modulo esterno. La web application comunicherà con l'EJB container attraverso chiamate remote.

Scendendo più nel dettaglio, ogni modulo può essere quindi composto dai seguenti EJB:

Remote Session EJB Il compito di questi bean è di comunicare con la web application che si occupa della presentation logic.

Local Session EJB In questi bean viene implementata la logica di business del modulo

Entity EJB Sono utilizzati per interagire con il database in caso sia presente.

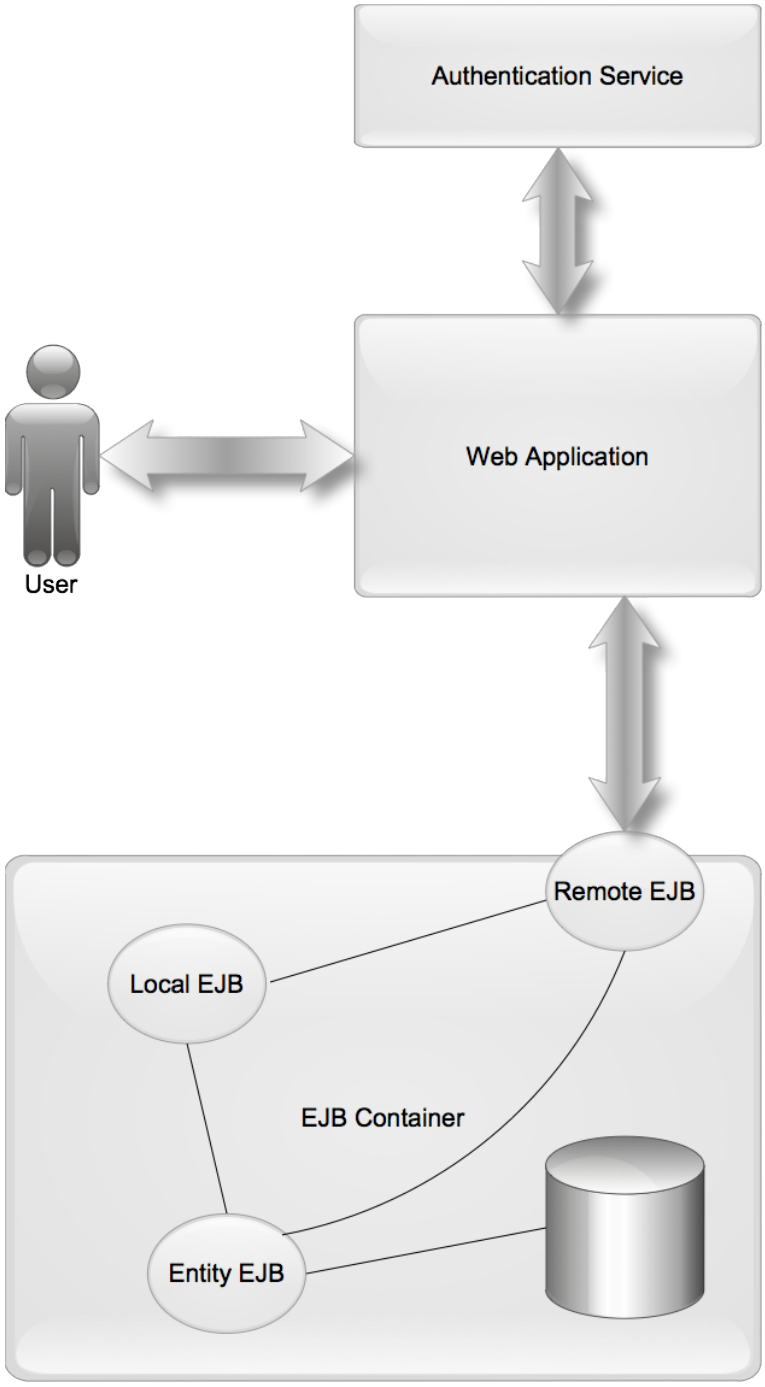

Figura 4.1: Architettura del sistema

Tramite questa architettura si riescono ad raggiungere alcuni obiettivi proposti, mentre altri saranno raggiunti attraverso una apposita architettura di deploy e l'integrazione di servizi esistenti.

Il business layer permetterà una facile estendibilità dell'applicazione in caso di futuri servizi esterni che vorranno accedere al database TCO. Inoltre agirà come middleware tra le applicazioni che vogliono accedere a questi dati e il database stesso, questo permette una ottima divisione dei compiti ed inoltre permette di fornire risposte totalmente indipendenti dell'EJB container anche di fronte a variazioni della struttura del database.

Il sistema in questo modo guadagnerà un certo livello di modularità e la sua estensione risulterà di semplice implementazione.

4.4 Deploy del sistema

Per esigenze di affidabilità e scalabilità ed inoltra per politica aziendale, tutti i componenti dell'applicazione (Figura 4.2) saranno messi in clustering, a partire dal web container, passando per l'EJB container fino ad arrivare al database server stesso e ai server di autenticazione. Questi ultimi due già presenti nella rete aziendale in tale configurazione. Tutti i componenti del sistema saranno raggiungibili per motivi di sicurezza soltanto attraverso la rete VPN globale che copre tutti gli uffici e siti produttivi del mondo. La gestione degli accessi come si può vedere sarà gestita tramite il servizio Active Directory presente globalmente.

Con questa configurazione non si avrà un *single point of failure* e a meno di problemi di routing e connettività all'interno della VPN si avrà garantito un sistema ad alta disponibilità.

In ogni caso si potrà procedere al fine tuning dell'architettura quando essa

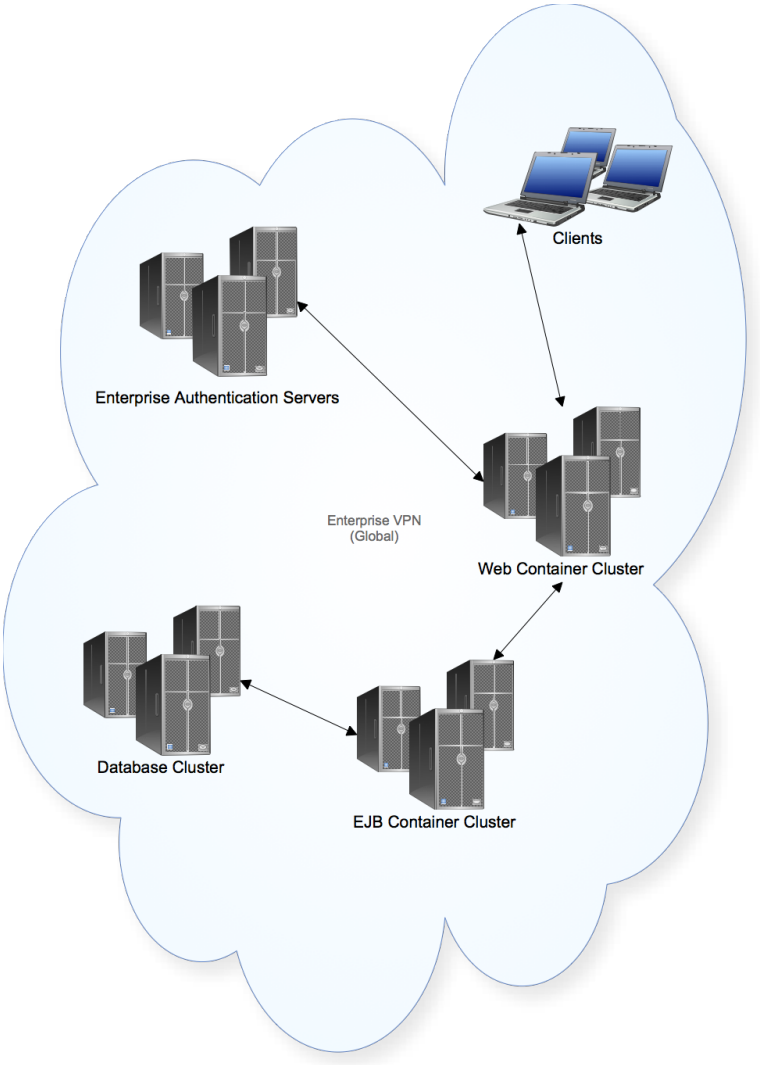

Figura 4.2: Esempio di deploy

sarà in produzione e sarà possibile valutare le performance in un contesto di produzione reale.

Capitolo 5

Implementazione

Dopo l'attenta analisi dei prototipi prodotti e la scelta della tecnologia Java EE 6, EJB per il Business Layer e GWT per il Presentation Layer siamo passati alla vera fase di progettazione ed implementazione.

Per fornire una più chiara spiegazione è necessario spiegare che con il termine *Applicazione* ci riferiremo alle entità primarie presenti nel database TCO e al quale sono connessi una serie di modelli (vedi Figura 5.2).

Per poter realizzare il prototipo è stato inoltre necessario individuare i casi d'uso del sistema, riportati in Figura 5.1. Sono principalmente presenti tre attori principali:

- *User*, i cui compiti saranno:
  - Accedere alle applicazioni presenti nel database
  - Eseguire azioni Create-Read-Update-Delete (CRUD)[1]
- *Application Manager*, i cui compiti saranno:
  - Eseguire operazioni Application wide, come calcoli ricorsivi sui sotto modelli per la distribuzione dei costi.

[1] http://en.wikipedia.org/wiki/Create,_read,_update_and_delete

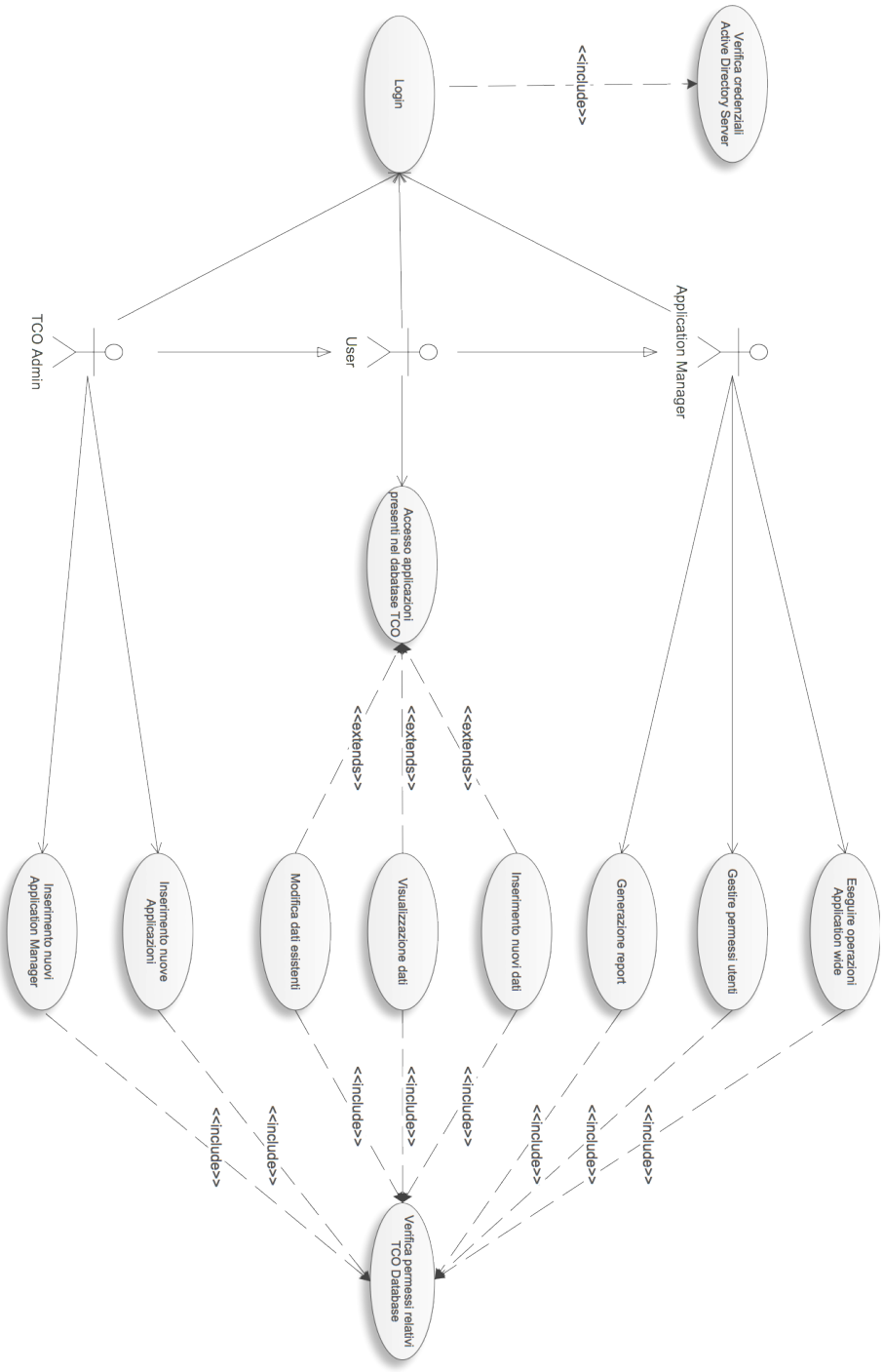

Figura 5.1: Diagramma dei casi d'uso

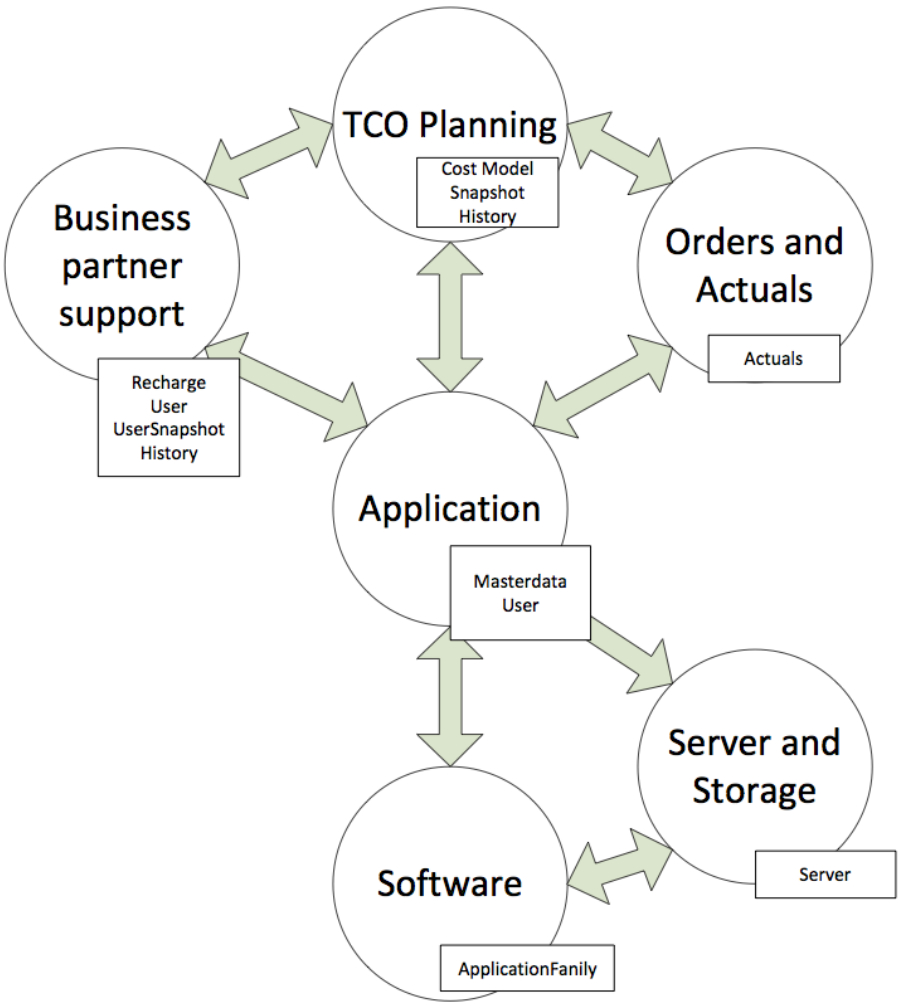

Figura 5.2: Aree funzionali modello TCO

  - Gestire permessi utenti.
  - Generare report relativi all'Applicazione.

- *TCO Administrator*, i cui compiti saranno:
  - Inserire nuove Applicazioni nel database.
  - Inserire nuovi Application Manager nel database e definirne i permessi.

Per ogni azione devono essere verificati i permessi, direttamente in una tabella del database, relativi all'utente che esegue l'operazione. Sono previsti diversi livelli di permesso, con diversa granularità. Dal permesso che autorizza un utente ad eseguire tutte le operazioni su di una Applicazione e sui modelli coinvolti, fino alla gestione per esempio della singola operazione di lettura su di un particolare attributo.

Come vedremo successivamente la gestione degli accessi sarà gestita a due livelli. Un primo livello in cui viene autenticato l'utente con le credenziali aziendali ed un secondo livello con un livello di capillarità maggiore, inerente alle azioni che l'utente può compiere sul singolo dato.

Nelle sezioni successive sarà illustrato in dettaglio il funzionamento della *business logic* e della *presentation logic* del prototipo, che realizzano le specifiche appena elencate.

5.1 Business Logic

Essendo il database TCO già presente (e già utilizzato attraverso una applicazione desktop minimale), è stato necessario creare gli EJB Entity (specifica JPA). Inizialmente si era tentato creare attraverso dei tool automatizzati, messi a disposizione dall'ambiente di sviluppo Eclipse, di creare gli EJB

Entity partendo dalle tabelle del database TCO. Questo però non ha portato ad un risultato soddisfacente, o per lo meno non completo, quindi è stato necessario un intervento manuale. Intervento che per lo più ha interessato il mapping delle varie relazioni tra i modelli, la gestione delle chiavi primarie composte attraverso classi Java marcate poi con l'annotazione *@Embeddable* [30].

Il compito generale della business logic è quello di regolare l'accesso ai dati presenti nel database TCO, fungere da middleware e quindi fornire un ulteriore livello di astrazione ed essere inoltre l'unico punto di accesso ai dati. Questo permetterà anche in ottica futura di essere la sola interfaccia con il database TCO, anche per successive applicazioni che dovranno accedere a tali dati. Quindi avrà anche la funzione di rendere disponibili e memorizzare dati da e verso la presentation logic, quindi la business logic sarà il punto di interscambio di dati tra il DBMS e la presentation logic.

Gli EJB presenti sul sistema saranno:

Entity Si occuperanno di realizzare l'ORM (Object-Relational Mapping) in modo da poter memorizzare i dati sul database.

Session Metteranno a disposizione i metodi remoti per permettere lo scambio dei dati con la presentation logic ed altre applicazioni future.

Per questi tipi di EJB saranno presenti entrambe le interfacce, remota e locale. Chiaramente visto che l'applicazione sarà totalmente distribuita la comunicazione tra la business logic e la presentation logic avverrà appunto tramite le interfacce remote.

Per quanto riguarda l'ORM chiaramente è stata utilizzata la specifica JPA e si è optato per l'implementazione *Hibernate*, in quanto poi in fase di deploy per la business logic la scelta dell'EJB Container è ricaduta su JBoss.

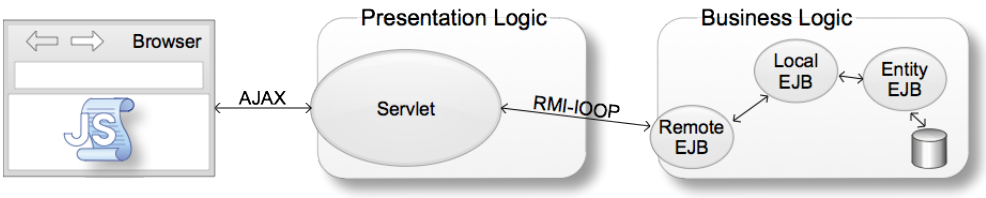

Figura 5.3: Architettura applicazione GWT

5.2 Presentation Logic

Illustrato il funzionamento della business logic è ora possibile descrivere la presentation logic. Come affermato nel capitolo 4 è stato deciso di utilizzare Google Web Toolkit (GWT) per implementare tale modulo, il cui funzionamento può essere così riepilogato (Figura 5.3):

- Tramite browser vengono visualizzate le pagine all'utente e viene eseguito il codice javascript generato dalla libreria.
- Il codice javascript, a seguito di un input da parte dell'utente, si occupa di effettuare le chiamate AJAX verso le implementazioni delle Servlet (sempre facenti parti della libreria), che a loro volta interrogano gli EJB remoti della business logic.
- Gli EJB eseguono le operazioni richieste e restituiscono i risultati alle Servlet che a loro volta li passano al browser, sempre tramite AJAX.

Come descritto precedentemente la presentation logic può essere installata su un cluster diverso rispetto a quello business logic. Per ottenere tale risultato è necessario prevedere un file di configurazione in cui sia possibile specificare dove si trova il cluster della business logic. Tale file è il *web.xml*, noto anche come descrittore di deployment, che è formalizzato nella specifica JavaEE e contiene tute le configurazioni della web application:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://
        java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    version="2.5" xmlns="http://java.sun.com/xml/ns/javaee">

<!-- Start: Config for Spring Security -->
    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.
            DelegatingFilterProxy</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <listener>
        <listener-class>org.springframework.web.context.
            ContextLoaderListener</listener-class>
    </listener>

<!-- End: Config for Spring Security -->

    <!-- Servlets -->
    <servlet>
        <servlet-name>ApplicationServiceImpl</servlet-name>
        <servlet-class>com.bombardier.transport.de.tco.
            presentationlayer.gwt.server.ApplicationServiceImpl</
            servlet-class>
    </servlet>
```

```
<servlet-mapping>
    <servlet-name>ApplicationServiceImpl</servlet-name>
    <url-pattern>/tcowebappgwt/applications</url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name>ApplicationManagerServiceImpl</servlet-name>
    <servlet-class>com.bombardier.transport.de.tco.
        presentationlayer.gwt.server.
        ApplicationManagerServiceImpl</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>ApplicationManagerServiceImpl</servlet-name>
    <url-pattern>/tcowebappgwt/ApplicationManagerService</url-
        pattern>
</servlet-mapping>

<servlet>
    <servlet-name>StatusServiceImpl</servlet-name>
    <servlet-class>com.bombardier.transport.de.tco.
        presentationlayer.gwt.server.StatusServiceImpl</servlet
        -class>
</servlet>
<servlet-mapping>
    <servlet-name>StatusServiceImpl</servlet-name>
    <url-pattern>/tcowebappgwt/StatusService</url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name>ApplicationFamilyServiceImpl</servlet-name>
    <servlet-class>com.bombardier.transport.de.tco.
        presentationlayer.gwt.server.
        ApplicationFamilyServiceImpl</servlet-class>
```

```
</servlet>
<servlet-mapping>
    <servlet-name>ApplicationFamilyServiceImpl</servlet-name>
    <url-pattern>/tcowebappgwt/ApplicationFamilyService</url-
        pattern>
</servlet-mapping>

<servlet>
    <servlet-name>CriticalityServiceImpl</servlet-name>
    <servlet-class>com.bombardier.transport.de.tco.
        presentationlayer.gwt.server.CriticalityServiceImpl</
        servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>CriticalityServiceImpl</servlet-name>
    <url-pattern>/tcowebappgwt/CriticalityService</url-pattern
        >
</servlet-mapping>

<servlet>
    <servlet-name>ChargingMethodServiceImpl</servlet-name>
    <servlet-class>com.bombardier.transport.de.tco.
        presentationlayer.gwt.server.ChargingMethodServiceImpl<
        /servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>ChargingMethodServiceImpl</servlet-name>
    <url-pattern>/tcowebappgwt/ChargingMethodService</url-
        pattern>
</servlet-mapping>

<servlet>
    <servlet-name>StaffServiceImpl</servlet-name>
```

```
    <servlet-class>com.bombardier.transport.de.tco.
        presentationlayer.gwt.server.StaffServiceImpl</servlet-
        class>
</servlet>
<servlet-mapping>
    <servlet-name>StaffServiceImpl</servlet-name>
    <url-pattern>/tcowebappgwt/StaffService</url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name>BudgetValueServiceImpl</servlet-name>
    <servlet-class>com.bombardier.transport.de.tco.
        presentationlayer.gwt.server.BudgetValueServiceImpl</
        servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>BudgetValueServiceImpl</servlet-name>
    <url-pattern>/tcowebappgwt/BudgetValueService</url-pattern
        >
</servlet-mapping>

<servlet>
    <servlet-name>FinPeriodServiceImpl</servlet-name>
    <servlet-class>com.bombardier.transport.de.tco.
        presentationlayer.gwt.server.FinPeriodServiceImpl</
        servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>FinPeriodServiceImpl</servlet-name>
    <url-pattern>/tcowebappgwt/FinPeriodService</url-pattern>
</servlet-mapping>

<servlet>
```

```
    <servlet-name>CompanyServiceImpl</servlet-name>
    <servlet-class>com.bombardier.transport.de.tco.
        presentationlayer.gwt.server.CompanyServiceImpl</
        servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>CompanyServiceImpl</servlet-name>
    <url-pattern>/tcowebappgwt/CompanyService</url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name>CosttypeServiceImpl</servlet-name>
    <servlet-class>com.bombardier.transport.de.tco.
        presentationlayer.gwt.server.CosttypeServiceImpl</
        servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>CosttypeServiceImpl</servlet-name>
    <url-pattern>/tcowebappgwt/CosttypeService</url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name>BudgetValueHistoryServiceImpl</servlet-name>
    <servlet-class>com.bombardier.transport.de.tco.
        presentationlayer.gwt.server.
        BudgetValueHistoryServiceImpl</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>BudgetValueHistoryServiceImpl</servlet-name>
    <url-pattern>/tcowebappgwt/BudgetValueHistoryService</url-
        pattern>
</servlet-mapping>
```

```
    <servlet>
        <servlet-name>MiddlewareServiceImpl</servlet-name>
        <servlet-class>com.bombardier.transport.de.tco.
            presentationlayer.gwt.server.MiddlewareServiceImpl</
            servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>MiddlewareServiceImpl</servlet-name>
        <url-pattern>/tcowebappgwt/MiddlewareService</url-pattern>
    </servlet-mapping>

    ...

    <!-- Default page to serve -->
    <welcome-file-list>
        <welcome-file>TcoWebAppGwt.jsp</welcome-file>
    </welcome-file-list>

</web-app>
```

Il descrittore di deployment sopra riportato non è esaustivo e riporta solo alcune delle tante servlet implementate.

5.3 Sicurezza

Per quanto riguarda la sicurezza dell'applicazione è stata gestita su due livelli. Un primo livello per moderare gli accessi all'applicazione stessa (Sezione 5.3.1) ed un secondo livello che verifica direttamente le operazioni che l'utente può eseguire sul singolo dato (Sezione 5.3.2).

5.3.1 Gestione degli accessi ed Autenticazione

Per quanto riguarda l'autenticazione dell'utente sull'applicazione uno dei requisiti è stato di dover utilizzare le credenziali già presenti nel sistema *Active Directory* che gestisce gli accessi globalmente per tutti i dipendenti dell'intero gruppo Bombardier. L'infrastruttura Active Directory si comporta in maniera totalmente trasparente, indipendentemente dalla sua dislocazione geografica. Da notare che Active Directory è un server Lightweight Directory Access Protocol (LDAP)[2] standard.

Per poter integrare le credenziali provenienti da Active Directory con l'applicazione si è utilizzato un modulo presente in un altro framework. Il modulo preso in considerazione è stato *Spring Security*[3]. Si tratta di un framework per la gestione degli accessi e dell'autenticazione molto personalizzabile ed è inoltre lo standard per garantire la sicurezza in applicazioni sviluppate con il framework Spring.

L'integrazione di esso non risulta molto problematica, ma è strettamente dipendente dalle necessità che si hanno. Nel caso dell'applicazione TCO è stata effettuata direttamente con un apposito file XML , *applicationContext.xml*, riportato di seguito.

```
<?xml version="1.0" encoding="UTF-8"?>

<beans:beans xmlns="http://www.springframework.org/schema/
    security"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/
        beans http://www.springframework.org/schema/beans/spring-
```

[2]RFC 4511, http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol

[3]http://static.springsource.org/spring-security/site/index.html

```
    beans.xsd http://www.springframework.org/schema/security
    http://www.springframework.org/schema/security/spring-
    security.xsd \http://www.springframework.org/schema/
    context http://www.springframework.org/schema/context">

<http auto-config="true">
    <intercept-url pattern="/login*" access="ROLE_ANONYMOUS" /
        >
    <intercept-url pattern="/*.jsp" access="
        IS_AUTHENTICATED_FULLY" />
    <intercept-url pattern="/*.html" access="
        IS_AUTHENTICATED_FULLY" />
    <intercept-url pattern="/TcoWebAppGwt/**"  access="
        IS_AUTHENTICATED_FULLY" />
 <form-login
    login-page="/login.jsp"
    default-target-url="/TcoWebAppGwt.jsp"
    authentication-failure-url = "/login.jsp?login_error=1"
    always-use-default-target="true" />
</http>

<beans:bean id="ldapAuthProvider"
    class="org.springframework.security.ldap.authentication.ad
        .ActiveDirectoryLdapAuthenticationProvider">
    <beans:constructor-arg value="" />
    <beans:constructor-arg value="ldap://xxxxxx.bombardier.
        com:3268/" />
    <beans:property name="convertSubErrorCodesToExceptions"
        value="true" />
</beans:bean>

<authentication-manager erase-credentials="true">
    <authentication-provider ref="ldapAuthProvider">
```

```
    </authentication-provider>
  </authentication-manager>

</beans:beans>
```

E' stato oscurato l'hostname del server Active Directory per motivi di sicurezza e privacy.

5.3.2 Operazioni sui dati

Per quanto riguarda la gestione della sicurezza sui dati stessi è stato seguito un altro approccio. Nel database TCO stesso è previsto una serie di relazioni tra tabelle che gestiscono i permessi relativi ai vari attori coinvolti. Ogni attore potrà eseguire solo un determinato set di operazioni su dati. Questi permessi saranno gestiti dall'Application Manager (o dal TCO Admin).

Il livello più capillare di permessi previsto è legato alle semplici azioni CRUD. Il livello di privilegio dell'utente viene sempre controllato prima di ogni operazione, poichè ad ogni azione chiamata sugli EJB Session viene effettuato un controllo di tale livello, passando l'identificativo dell'utente, ed in caso di errore viene generate un eccezione. Più difficoltoso è stato considerare la gestione di questo livello di accesso sulla presentation logic (vedere Sezione 5.4).

5.3.3 DTO - Trasferimento dati tra layers

Per quanto riguarda il trasferimento dei dati tra la business logic e la presentation logic sono stati rinvenuti dei problemi di serializzazione.

Il problema riscontrato è relativo ad eccezioni del tipo *SerializationException* dovute al fatto che la business logic cercava di serializzare degli oggetti che includevano tipi di dato non serializzabili.

La definizione di *serializzabile* qui sta a significare che il meccanismo GWT RPC sa come serializzare e deserializzare il tipo da bytecode a JSON e viceversa. Per dichiarare un tipo serializzabile, è possibile implementare l'interfaccia *IsSerializable*, creata appositamente per questo scopo, o implementare l'interfaccia *java.io.Serializable*, il tutto a condizione che i membri della classe e i suoi metodi consistano di tipi che sono anch'essi serializzabili.

Nel caso degli oggetti Hibernate era stata implementate l'interfaccia *Serializable*, ma questo non ha portato a nessun risultato.

La motivazione di tutto questo è legata al fatto che quando viene prelevato un oggetto per essere trasformato in un oggetto Hibernate, esso viene arricchito di diverse caratteristiche per essere reso *persistente*. Ovvero la libreria *Javassist* effettivamente riscrive il bytecode e sostituisce l'oggetto per permettere il corretto funzionamento di Hibernate. Questo per GWT RPC sta a significare che quando l'oggetto sta per essere serializzato in realtà non è più lo stesso oggetto che il compilatore si aspetta di trovare al momento della deserializzazione.

Una possibile soluzione sarebbe quella di sostituire i tipi di dato ancora una volta nella direzione opposta prima di ritornare attraverso la chiamata RPC lato server. Questo è fattibile, e risolverebbe il problema che abbiamo incontrato, ma sono presenti altre problematiche connesse. Uno dei grossi vantaggi di Hibernate è quello di caricare associazioni tra oggetti e qui, in caso di *Lazy Association* si incorrerebbe in altri problemi, come per esempio eccezioni del tipo *LazyInitializationException*.

Ci sono varie soluzioni per ovviare a questo problema:

- *Data Transfer Objects*: creare dei *light object* che rappresenteranno i dati sul lato client. Ovvero dei semplici Plain Old Java Object (POJO) utilizzati per visualizzare i dati client-side.

- *Dozer*: è una libreria open source che permette di generare DTO a partire da descrizioni XML.

- *Gilead* (Hibernate4Gwt): è anch'esso una libreria open source. Implementa una soluzione più trasparente per trasferire dati tra Hibernate e GWT.

- altre soluzioni minori.

Ognuna di queste soluzioni presenta i propri vantaggi e svantaggi, che variano soprattutto per quanto riguarda l'onere per lo sviluppatore. Tuttavia, la preoccupazione generale attraverso ciascuna di queste strategie, così come altri aspetti dello sviluppo di applicazioni web, deve essere sempre la valutazione delle prestazioni, poichè esse influiscono in maniera diretta sugli utenti. Un certo numero di questi approcci a volte può portare un notevole sovraccarico a runtime. Ad esempio, l'approccio Dozer e Gilead può diventare faticoso per la UX quando ci sono grandi insiemi di dati da serializzare, mentre la soluzione con DTO può essere progettata per essere la più coincisa ed efficace e migliorare le prestazioni.

La soluzione di utilizzare DTO, come degli oggetti proxy tra il presentation layer e il business layer è risultata la preferita, nonostante essa prevedesse una considerevole mole di lavoro in più, poichè è stato necessario creare manualmente tutti i DTO. Questo ha permesso di eliminare il problema della serializzazione tra oggetti Hibernate e la presentation logic ed ha inoltre introdotto un ulteriore livello di astrazione che può in ogni caso essere sfruttato per sviluppi futuri.

La scelta della soluzione con DTO è stata percorsa anche perchè si è preferito non legarsi a delle librerie esterne che al momento dello sviluppo non presentavano un grosso supporto alle spalle. Si è voluto quindi evitare,

in ottica futura, di appoggiarsi a dei progetti che potessero essere non più manutenuti e quindi causare problemi con sviluppi futuri.

5.4 Altre problematiche affrontate

Sono state riscontrate altre problematiche minori, specifiche di particolari richieste dell'azienda.

Se ne elencano alcune:

- Compatibilità con Internet Explorer 8: software standard presente nei computer dei dipendenti.
- Design dell'interfaccia utente: è stato fatto uno studio minimale dell'interfaccia utente poichè uno dei requisiti non direttamente espressi era quello appunto di avere una facilità di utilizzo dell'applicazione anche per utenti meno abituati all'utilizzo di computer. Questo perchè l'utilizzo di tale applicazione coinvolge personale che è direttamente addetto alla gestione di aspetti tecnici e che fa un utilizzo odierno della tecnologia, ma anche di dipendenti di altri funzioni aziendali più focalizzati ad aspetti di *finance* che avevano riscontrato notevoli problemi di utilizzo con il software precedente.
- Suggerimenti su campo a inserimento libero con moli di dati elevate: questo ha portato alla creazione di una soluzione ad hoc della *SuggestBox* già presente in GWT (conosciuta anche come *Oracolo*), per ovviare ai notevoli problemi di prestazioni che si verificavano con la versione standard.

5.5 Deploy

Per quanto riguarda il deploy dell'applicazione sono state effettuate le seguenti scelte:

- JBoss AS 7.1 come EJB Container
- Tomcat 7 come Web Container per l'applicazione GWT
- SQL Server 2008 per il RDBMS, questa scelta è stata obbligata in quanto il database TCO già presente su questa piattaforma.

Per quanto riguarda ogni singolo punto è stato provveduto al clustering dove esso non fosse già presente.

5.6 Metodologia di sviluppo

Per quanto riguarda la metodologia di sviluppo utilizzata si è deciso di seguire una metodologia Agile, in particolar modo *Scrum*.

Visto l'alto numero di persone con diversi requisiti da dover soddisfare, avere un approccio iterativo con dei rilasci frequenti (era previsto un *deliverable* ogni settimana) è stato di grande aiuto per modificare in corso d'opera molti dettagli implementativi che non erano emersi da una prima analisi.

L'utilizzo di un approccio *Waterfall*, anche se la dimensione del progetto non è molto grande avrebbe potuto causare notevoli problemi nel momento in cui si fosse arrivati ad una fase di rilascio finale con dei requisiti che per loro natura erano poco chiari e mal definiti. Inoltre all'interno dell'azienda era già stato sperimentato l'utilizzo di Scrum in altri progetti e quindi l'adozione di esso anche per lo sviluppo di questa applicazione è risultato in una scelta naturale ed ottimale.

Capitolo 6

Conclusioni

Nell'ambito della tesi è stato progettato e realizzato un sistema per la gestione dei dati provenienti da un database di notevoli dimensioni che raccoglie tutte le informazioni necessarie all'approccio Total Cost of Ownership sviluppato dalla multinazionale Garter Inc., leader mondiale nella consulenza strategica, ricerca e analisi nel campo dell'Information Technology. Il compito dell'approccio TCO è quello di calcolare tutti i costi del ciclo di vita di un'apparecchiatura informatica IT, per l'acquisto, l'installazione, la gestione, la manutenzione e il suo smantellamento.

L'approccio TCO è basato sulla considerazione che il costo totale di utilizzo di una apparecchiatura IT non dipende solo dai costi di acquisto, ma anche dai tutti i costi che intervengono durante l'intera vita di esercizio dello strumento. I costi connessi alla stessa formulazione del TCO, che rientrano allo stesso modo nel TCO sono abbastanza rilevanti, dunque tale metodo viene spesso utilizzato solo da grandi aziende, in cui l'IT rappresenta una variabile strategica rilevante. Nella fase iniziale della tesi è stato dunque compiuto uno studio atto ad individuare quali fossero i requisiti del sistema e le tecnologie più adatte a realizzarlo. Da questo studio è quindi emersa la

scelta di utilizzare Java Enterprise Edition v6 per la parte di business logic e Google Web Toolkit per la parte di presentation logic.

Dopo un attenta analisi del modello, si è passati ad analizzare i requisiti che un applicazione per la gestione di tale approccio in un ambiente enterprise deve avere.

Successivamente si è passati alle fasi di progettazione e di sviluppo dell'applicazione Java Enterprise, tramite cui è stato possibile verificare come, tale tecnologia, permetta di soddisfare le sempre maggiori esigenze aziendali di avere a disposizione applicazioni modulari, scalabili, affidabili e robuste. E' stato infatti dimostrato come l'architettura che è stata proposta rispecchi appieno i requisiti, permettendo una perfetta sinergia tra i sistemi già presenti ed una facile integrazione dei sistemi che saranno sviluppati in futuro.

Uno dei maggiori punti di forza ottenuti con la soluzione implementata è stata la totale astrazione dei dati presenti sul database dalla presentation logic. Poichè la struttura del database TCO cambia, anche se di poco, piuttosto frequentemente, questo aiuterà a mantenere un comportamento coerente ed a permettere una più facile manutenibilità dell'applicazione e un suo futuro ampliamento. L'utilizzo di un architettura modulare, dove la business logic è totalmente divisa dalla presentation logic permetterà di poter sviluppare ulteriori applicazioni, anche completamente indipendenti, che potranno accedere a questo layer e ottenere dati utili ad altri sistemi, come per esempio il sistema ERP presente in azienda.

L'utilizzo di un'architettura modulare ha quindi portato dei notevoli vantaggi, ogni modulo può difatti essere eseguito su di un apposito cluster, permettendo di allocare le risorse a disposizione in funzione delle necessità computazionali di ogni modulo.

L'architettura presentata più che dover gestire un alto livello di acces-

si dovrà garantire l'alta disponibilità del sistema e una elevata affidabilità, poichè il database TCO risulta essere di vitale importanza nell'utilizzo di ogni giorno e i dati contenuti al suo interno consentono di avere un resoconto dettagliato dei costi inerenti alle apparecchiature., requisito essenziale per una multinazionale delle dimensioni di Bombardier, dove l'utilizzo delle apparecchiature IT è una necessità.

L'utilizzo di gruppi di cluster ha quindi permesso di ottenere un'applicazione affidabile e robusta, difatti in caso di *failure* di un server il sistema sarà soggetto solamente ad un calo prestazionale.

In ultima nota è necessario soffermarsi su come lo sviluppo di un'interfaccia per gli utenti realizzata completamente tramite tecnologia AJAX, grazie all'utilizzo di Google Web Toolkit, abbia rappresentato un notevole valore aggiunto per l'applicazione. Questo ha permesso di avere un applicazione altamente interattiva, che fornisce una elevata esperienza utente. Lo studio dell'interfaccia ha inoltre permesso di creare un applicazione con un alto livello di usabilità, requisito essenziale dovuto alla elevata eterogeneità degli utenti.

Appendice A

Glossario

B2B Business to Business

B2C Business to Consumer

BDD Behavior Driven Development

CDN Content Delivery Network

CRUD Create-Read-Update-Delete

DTO Data Transfer Object

EJB Enterprise JavaBeans

ERTMS/ETCS European Rail Traffic Management System/European Train Control System

JAAS Java Authentication and Authorization Service

JavaEE Java Enterprise Edition

JCE Java Cryptography Extension

JNDI Java Naming and Directory Interface

JPA Java Peristence API

LDAP Lightweight Directory Access Protocol

MVC Model-View-Controller

ORM Object-Relational Mapping

POJO Plain Old Java Object

RDBMS Relational Database Management System

RPC Remote Procedure Call

TCO Total Cost of Ownership

TDD Test Driven Development

Ringraziamenti

Questo è di gran lunga il capitolo più difficile da scrivere. Un altro punto di arrivo, un altro traguardo raggiunto. Vorrei ringraziare una ad una tutte le persone che mi hanno accompagnato fino a qui, chi ha fatto tutto il percorso con me, chi solo l'inizio, chi solo la fine. Per primi devo ringraziare i miei genitori, Carla e Marcello, che anche in questa seconda avventura universitaria mi hanno dato tutto il loro supporto, credendo in me anche quando era difficile farlo.

Vorrei ringraziare tutti gli amici, i non amici e i conoscenti che in qualsiasi modo hanno contribuito, anche con un sorriso o una critica, a raggiungere questo giorno.

Guardandomi indietro vedo tante cose fatte, tante cose mancate, tanti successi e altrettanti errori, ecco li ripeterei tutti, dal primo all'ultimo. La mia vita dai 19 ad adesso è stata per certi versi piuttosto movimentata. Ho cercato sempre di dare spazio a tutto, quando il tempo era sufficiente forse a fare la metà delle cose che avevo in mente. Non ho mai accettato di dover dare priorità ad una sola cosa, compiendo errori ed essendo quasi sempre in ritardo sui miei obiettivi. Spero che da tutte queste esperienze abbia imparato qualcosa e che in futuro la vita vissuta mi porti consiglio.

Nell'ultimo anno ho fatto uno passo importante, ho lasciato l'Italia. Lasciare la famiglia, gli amici non è facile, ma ho trovato un mondo pieno di

opportunità, opportunità che sono difficili adesso da rifiutare. Ho anche incontrato qua una persona speciale, Maryem, che mi ha supportato in questi ultimi difficili mesi e spero che continui a farlo per molto a lungo ancora.

Voglio fare un ringraziamento speciale a *Casa Brennero* ed in particolar modo (in ordine sparso), Andrea, Francesco e Felice, con cui ho condiviso dei momenti fantastici, ragazzi non li dimenticherò mai. Ancora una volta c'è una persona che merita un ringraziamento immenso, Michele, amico, compagno di studi e grande supporto in tutti i momenti della mia vita, avrò tanto da fare per sdebitarmi. Voglio dire un grande grazie anche ad Alessio, Mirko, Diego e Marcello, miei compagni nei progetti più impegnativi, che hanno condiviso con me lunghe nottate e hanno subito i miei eccessi nel perfezionare qualcosa.

Vorrei rendere questa lista infinita, ma non ne ho ne il tempo ne lo spazio. Mi scuso anticipatamente con tutti coloro che si meriterebbero di avere un personale ringraziamento in questa pagina e che mi sono dimenticato. Anche questa volta, come in diversi momenti della mia vita, mi sto trovando a correre per rispettare una deadline, quindi penso che ancora di strada per migliorare ne dovrò fare tanta.

Con affetto

Michele

Bibliografia

[1] Bombardier. [Online]. Available: http://en.wikipedia.org/wiki/Bombardier_Inc.

[2] B. Gomolski, J. Grigg, and K. Potter, "It spending and staffing survey results," *Gartner Group*, 2001.

[3] Java EE v6. [Online]. Available: http://www.oracle.com/technetwork/java/javaee/overview/index.html

[4] GlassFish. [Online]. Available: https://glassfish.dev.java.net/

[5] JBoss Application Server. [Online]. Available: http://jboss.org/

[6] WebSphere. [Online]. Available: http://www-01.ibm.com/software/websphere/

[7] Oracle WebLogic Server. [Online]. Available: http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html

[8] Enterprise JavaBeans Technology. [Online]. Available: http://java.sun.com/products/ejb/

[9] Java Persistence API. [Online]. Available: http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html

[10] Java Message Service. [Online]. Available: http://www.oracle.com/technetwork/java/index-jsp-142945.html

[11] Java Cryptography Extension. [Online]. Available: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html

[12] Java Authentication and Authorization Service. [Online]. Available: http://download.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html

[13] Java RMI over IIOP. [Online]. Available: http://java.sun.com/products/rmi-iiop/

[14] CORBA/IIOP Specifications. [Online]. Available: http://www.omg.org/technology/documents/corba_spec_catalog.htm

[15] Java Annotations. [Online]. Available: http://download.oracle.com/javase/1.5.0/docs/guide/language/annotations.html

[16] Java Naming and Directory Interface. [Online]. Available: http://www.oracle.com/technetwork/java/index-jsp-137536.html

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns*. Addison-Wesley Reading, MA, 2002.

[18] Java Servlet Technology. [Online]. Available: http://www.oracle.com/technetwork/java/index-jsp-135475.html

[19] JavaServer Pages. [Online]. Available: http://java.sun.com/products/jsp/

[20] JavaServer Faces Technology. [Online]. Available: http://java.sun.com/javaee/javaserverfaces/

[21] Google Web Toolkit. [Online]. Available: http://code.google.com/intl/it-IT/webtoolkit/

[22] JUnit. [Online]. Available: http://www.junit.org/

[23] R. Sriganesh, G. Brose, and M. Silverman, *Mastering Enterprise JavaBeans 3.0*. John Wiley & Sons, Inc. New York, NY, USA, 2006.

[24] The Seam Framework. [Online]. Available: http://seamframework.org/

[25] D. Allen, *Seam in Action*. Manning Publications Co. Greenwich, CT, USA, 2008.

[26] R. Johnson, *Expert One-on-One Java EE Design and Development*. Wrox Press, 2002.

[27] I. Terracotta and T. Inc, *The Definitive Guide to Terracotta: Cluster the JVM for Spring, Hibernate and POJO Scalability*. Springer, 2008.

[28] Enterprise JavaBeans 3.1. [Online]. Available: http://jcp.org/aboutJava/communityprocess/final/jsr318/index.html

[29] T. O'reilly, "What is Web 2.0: Design patterns and business models for the next generation of software," *O'Reilly Media*, 2005.

[30] B. Burke and R. Monson-Haefel, *Enterprise JavaBeans 3.0 (5th Edition)*. O'Reilly Media, Inc., 2006.